ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

PROYECTO ARDUINO UPNA

Aitor Iriarte Pastor y Carlos Zuazu Ayesa

Alfredo Pina y Vicente Senosiain

Pamplona, 25 de enero del 2011

Índice

1. Objetivo

El objetivo del proyecto que se va a realizar es diseñar e implementar una serie de pequeños sub-sistemas robóticos con distintas funcionalidades, con la intención de formar una base de componentes hardware y software para futuros proyectos que utilicen la tecnología ARDUINO, de manera que se pueda utilizar este conocimiento para participar en distintas competiciones de robótica.

Descripción del PFC

Inicialmente se ha hecho un estudio sobre distintos concursos de robótica con el fin de ver que necesidades presentan los mismos, tales como tipos de competiciones, materiales utilizados y que placas controladoras son las más utilizadas. Hemos comprobado como depende mucho de cada concurso y de cada edición el tipo de requerimientos que necesita un robot para competir, siendo los tipos de competiciones más habituales los siguientes:

- Velocistas: El concurso de velocidad consistirá en una carrera de persecución entre dos µRs en una pista cerrada, comenzando en puntos opuestos y avanzando en el mismo sentido (la pista será simétrica respecto a dos ejes garantizando que ambos µRs encuentran tramos de pista similares en su recorrido). El objeto, por tanto, será la consecución de altas velocidades de marcha en un recorrido perfectamente preestablecido.

- Rastreadores: Esta prueba valorará la habilidad de un Robot para recorrer un camino sinuoso, previamente establecido, en el menor tiempo posible. Se conjugarán, por lo tanto, dos aspectos importantes: capacidad de detección y seguimiento del camino y rapidez con que se ejecutan las maniobras.

- Sumos: Dos robots, emulando a los luchadores de Sumo, intentarán arrojar a su oponente del Área de Combate cumpliendo ciertas reglas detalladas en esta normativa.

- Recolectores: Uno o dos robots compiten en un tablero específico para ver quien recolecta y/o clasifica distintos objetos que en cada concurso representan distintos elementos.

- Soccer: En la prueba de Soccer de la RCJ los estudiantes tendrán que diseñar, construir y programar dos robots que sean capaces de jugar al fútbol contra otro par de robots oponentes. El campo de juego tiene el tamaño aproximado de una mesa de ping-pong sobre el que hay impreso un degradado de grises para ayudar a posicionar a los robots. Las paredes son altas de forma que impiden que ningún robot se pueda salir del campo. Para poder localizar con facilidad la pelota esta emite luz infrarroja.

Hoy en día existen distintos concursos por todo el mundo. Para nosotros los concursos que nos tocan más cerca son los de nivel nacional, o nivel internacional a nivel europeo. Algunos de estos concursos necesitan una fase de clasificación previa, tal y como se especifica a continuación. El listado de concursos y sus fechas se encuentra bien detallado en la siguiente página http://www.roboticadeservicios.com/eventos.html. Notros añadimos algunos de estos concursos para entender a grandes rasgos en que consisten.

A nivel nacional:

- Alcabot: Organizado en la Escuela Politécnica de la Univ. de Alcalá (Madrid), con sus tradicionales pruebas de velocistas, sumo, rastreadores, laberinto, además de ser clasificatorio para la RoboCup y el Eurobot. Incluye charlas, conferencias y distintos talleres.

- RoboCup Junior Open Spain 2010: Competición clasificatoria de España para el Campeonato del mundo de la RoboCup Junior - UAH - Alcalá de Henares (Madrid)

- FIRST LEGO League España: Final española de esta competición internacional utilizando Kits LEGO de robótica. Previamente se disputarán torneos en Jaén, Asturias, Barcelona, Castilla y León, País Vasco, Gerona, Madrid, etc.

- MadridBot: Concurso organizado por centros que imparten las enseñanzas del Ciclo Formativo de Grado Superior de Desarrollo de Productos Electrónicos en la Comunidad Autónoma de Madrid. Pruebas de rastreadores, velocistas, laberinto y prueba libre.

A nivel internacional:

- Eurobot: Este es el concurso de robótica mas importante de Europa y donde se compite tras haber superado las rondas clasificatorias de cada país. Cada año se presenta un nuevo reto que deben afrontar todos los concursantes siendo los objetivos de los años anteriores tales como jugar a rugby, bolos, clasificar residuos, recoger muestras en Marte y realizar una recolección de alimentos. Este año 2011 se plantea un juego parecido al ajedrez con algunas reglas especiales.

- Robocup Junior (Singapur): Campeonato mundial de robótica con clasificatorios por todo el mundo con competiciones de sumo, soccer, rastreadores, rescatadores, velocistas, etc.

- RoboParty (Portugal): Jornadas de robótica organizadas por la Universidad do Mihno. Hay ponencias, demostraciones de robots y competición de robótica con pruebas de robots seguidores, prueba de obstáculos y prueba de danza.

Tras haber estudiado en profundidad dicho concurso, hemos decidido realizar una serie de pequeños "hitos" que queremos que realice un robot, basándonos en la placa Arduino y en el microprocesador ATMega que es el más utilizado y recomendado para todos estos concursos anteriormente mencionados, y que permita preparar un robot para este tipo de competiciones.
Arduino es una plataforma de hardware libre basada en una sencilla placa de entradas y salidas simples y un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring. Arduino se puede utilizar para desarrollar objetos interactivos autónomos o puede ser conectado a software del ordenador (por ejemplo: Macromedia Flash, Processing, Max/MSP, Pure Data). Las placas se pueden montar a mano o adquirirse. El entorno de desarrollo integrado libre se puede descargar gratuitamente.
Las plataformas Arduino están basadas en los microcontroladores Atmega168, Atmega328, Atmega1280, ATmega8 y otros similares, chips sencillos y de bajo coste que permiten el desarrollo de múltiples diseños. Al ser open-hardware, tanto su diseño como su distribución son libres. Es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia. El proyecto Arduino recibió una mención honorífica en la categoría de Comunidades Digital en el Prix Ars Electronica de 2006. Las distintas estrategias de programación deberán tener en cuenta que el programa debe ser lo más simple y rápido en ejecución posible, siendo la precisión un aspecto muy importante.

Con esto pretendemos crear "know-how" sobre esta temática para futuros proyectistas en el que tendremos la información necesaria para construir distintas partes de un robot, y adaptarlas a las necesidades que requiera cada proyecto. De esta manera conseguimos que solo tengan que fijarse en mejorar el diseño, en adaptarlo a las diferentes competiciones y que por lo tanto no tengan que empezar desde cero.

Este proyecto se centra en la parte Hardware de la realización o creación de estas partes, y además se encargará de la parte Software. La parte de Software estará totalmente ligada a lo que se realizará en la parte Hardware.

Este proyecto se estructura de la siguiente manera:

1. Investigación:
   - Acerca de las distintas competiciones de robótica existentes.
   - Acerca del hardware Arduino.
   - Componentes (sensores, motores, displays,…).
   - Ejemplos de robots ya realizados.
2. Desarrollo:
   - Implementación de pequeños ejemplos simples.
   - Preparación de distintos hitos (que se corresponden a diferentes subsistemas):
     a. Movimiento de motores para desplazamiento.
     b. Detección de objetos a través de sensores.
     c. Esquivar objetos detectados.
     d. Seguimiento de líneas. Detección de contrastes.
     e. Detección de la distancia a objetos.
     f. Movimientos realizados por pulsación y grabación de entradas.
3. Realización de la memoria del PFC.
4. Realización de un Blog.
5. Presentación y defensa del PFC.

2. Introducción

2.1 Justificación del proyecto

Es difícil pensar que en la sociedad de hoy en día a pesar de la cantidad de recursos tecnológicos que poseemos quede algún servicio público incompleto, deficiente o simplemente que no cumpla con creces los objetivos para los que ha sido creado. Nada, más lejos de la realidad, pues si miramos a nuestro alrededor podemos encontrar multitud de servicios que bien por falta de estudio y acondicionamiento o bien por aspectos económicos, no sólo no cumplen con los cánones exigidos sino que lejos de facilitarnos las cosas nos las complican aún más.
Nosotros hemos decidido centrar el estudio en la elaboración de distintos robots que utilicen una serie de componentes que sean útiles para la realización de robots más complejos y competitivos. Esto, creemos, que puede ser una estupenda base para abrir una vía de proyectos relacionados con la robótica con la finalidad de crear un grupo de trabajo que se vaya alimentando de los PFC de los alumnos de la UPNA, y poder competir a nivel nacional e internacional en representación de la misma. Por supuesto esto sería una muy buena propaganda para la universidad otorgándole un prestigio en este campo de investigación.

2.2 Antecedentes

En la actualidad existen una gran variedad de robots que realizan funciones similares a las que deberán realizar nuestros robots, siendo alguno de ellos creado exclusivamente para que desempeñe la función para la cual nosotros lo hemos creado. Aún así hemos intentado mostrar un producto comercial con un buen acabado final para poder comprobar cuales son nuestras limitaciones.

De todos los robots que hemos encontrado, hemos querido mostrar algunos ejemplos de ellos con características similares a nuestros robots; así como ejemplos con prestaciones superiores a modo de ampliación de conocimientos y con la finalidad de ver unos productos de utilidad real.

2.2.1 Seguidor de líneas Pololu 3pi

Robot seguidor de líneas programable que incorpora un chip AVR Atmega328, por lo que es compatible con el entorno de desarrollo de Arduino.

Robot seguidor de líneas programable que incorpora un chip AVR Atmega328, por lo que es compatible con el entorno de desarrollo de Arduino, 5 sensores de reflexión, pulsadores, un pequeño LCD de 8x2 caracteres para mostrar mensajes a gusto del usuario junto con tres pulsadores, un buzzer para emitir sonidos y dos motores con reductoras lo que le permite alcanzar una velocidad de cerca de 1 metro por segundo. Todo esto en tan sólo un chasis circular de 9,7cm de diámetro.

- **Microcontrolador:** ATmega328
- **Diámetro del chasis:** 9,7cm
- **Driver de motores:**TB6612FNG
- **Motores:** 2
- **Interfaces I/O disponibles para el usuario:** 21

- **Voltaje mínimo:** 3 Vcc
- **Voltaje máximo:** 7 Vcc
- **Frecuencia PWM máxima:** 80 kHz
- Protección contra polaridad invertida incluida

Éste es un robot muy divertido con un montón de potencial. Fácil de utilizar ya que podemos probarlo desde los primeros minutos que lo tengamos en las manos y además dispone de muchísima documentación, por lo que es altamente flexible si queremos meterle mano.

Se recomienda utilizar un programador externo como el Orangutan USB.

Viene precargado con un programa de ejemplo con el cual podemos hacer girar los motores con la ayuda de los botones en placa, reproducir una melodía, ver gráficamente en la pantalla LCD el estado de los sensores etc.

También está disponible un kit de expansión para aumentar las funcionalidades del robot (ver productos relacionados).

Documentación:

- Pololu AVR C/C++ Library User's Guide
- Manuel de uso del programador Orangutan y Robot 3pi con el entorno de desarrollo de Arduino
- Guía de referencia de la librería AVR de Pololu para AVR
- Cómo realizar carreras de seguidores de líneas y resolver laberintos con el robot

Manuales y librerías de referencia:

- Robot 3pi Guía de usuario y esquema (Original - Ingles) (66k pdf)
- Robot 3pi guía de usuario (Español) (2483k pdf)
- Esquema simplificado (40k pdf)
- Liberia AVR de Pololu (Binario) (254k zip)
- Librería AVR de Pololu (Código Fuente) (295k zip)
- Programa de ejemplo (archivo .hex)(29k hex)
- Datasheet Toshiba TB6612FNG motor driver datasheet (207k pdf)

Precio: 79,90 € (+iva)

2.2.2 Roomba 520

Los aspiradores iRobot Roomba son máquinas inteligentes capaces de limpiar el suelo sin ningún tipo de supervisión. Gracias a sus sensores y a su inteligencia artificial, Roomba es capaz de evaluar su entorno hasta 50 veces por segundo y modificar su comportamiento para adaptarse a él.
Además de inteligencia, Roomba dispone de un sistema de limpieza en 3 fases que le permite recoger todo tipo de suciedad, pelo, polvo y pequeños objetos y depositarlos en un depósito o en su filtro.

Roomba es capaz de limpiar debajo de los muebles

El **modelo 520** se caracteriza por su capacidad para limpiar hasta 3 habitaciones con una sola carga, asegurando una limpieza efectiva con un consumo mínimo.

Entre sus características destaca la auto-recarga, es decir, que al acabar la tarea de limpieza, regresa a su base de carga para recargar la batería y estar listo para la siguiente vez que vuelvas a utilizarlo.

¡Se carga ella sola!

Características del modelo 520:

- Nuevo diseño modular, sólido y resistente para aumentar la durabilidad de la máquina.
- Tecnología anti-enredos que permite que Roomba pueda liberarse de los flecos o cordones que hayan quedado enredados.
- Sensores inteligentes Light-touch, que permiten reducir la velocidad de la máquina cuando se aproxima a un mueble u objeto, con la finalidad de impactar de manera suave y silenciosa.
- Incorpora un solo botón de limpieza “Clean”.
- Nuevo sistema de movimiento que permite a Roomba andar en alfombras más gruesas, subir desniveles mayores y pasar de una superficie a otra con facilidad.
- Nuevo diseño del recipiente que permite recoger más suciedad.
- Sistema reforzado de rodillos y cepillos laterales para realizar una limpieza en mayor profundidad y conseguir los mejores resultados.

Accesorios incluidos en iRobot Roomba 520:

- Base de carga.
- Batería APS.
- Cargador rápido de 3 horas.

Características generales:

- Medidas: 33,65 cm. de diámetro x 7,03 cm. de alto.
- Peso: 3.6 Kg. (con batería).
- Bolsas aspiradora: No necesita bolsa
- Garantía de 2 años.
- Caja y embalaje realizados con material 100% reciclable

Precio: 249 € (+iva)

2.2.3 Bee-Bot:

Ganador del Premio del robot programable de suelo, el diseño amigable y sencillo del Bee-Bot es un perfecto siendo un punto de partida para la enseñanza de control, de dirección y lenguaje de programación para niños pequeños

Descripción:

• Diseño robusto y pequeño
• Botones claros y brillantes
• Memoria de hasta 40 pasos
• Fácil de programar
• Los sonidos y los ojos brillantes confirmar sus instrucciones
• Fantástico recursos transversales disponibles
• Se mueve con precisión con pasos de 15cm y giros de 90 °

Precio: 63,95 € (+iva)

2.3 Objetivos

Después de comentar los antecedentes de lo que denominamos "robots automáticos", y de relatar algunos ejemplos de robots con características similares a los nuestros, vamos a introducir los objetivos que deberán alcanzar los robots presentados en este proyecto.

a. Movimiento de motores para desplazamiento.
b. Detección de objetos a través de sensores.
c. Esquivar objetos detectados.
d. Seguimiento de líneas. Detección de contrastes.
e. Detección de la distancia a objetos.
f. Movimientos realizados por pulsación y grabación de entradas.

2.4 Alcance del proyecto.

Para alcanzar los objetivos marcados en la realización de este proyecto hemos tenido que estudiar el sistema de concursos implantado en nuestro país y en el extranjero para poder hacernos una idea de las características de los mismos en cuanto a cumplir con las premisas para las cuales había sido designado. Una vez llegados a este punto, estudiamos la manera de cómo complementar dicho sistema a través de un medio tecnológico que permitiera la creación de una plataforma autónoma que ofreciera soluciones a dichas características de una forma lo más sencilla posible.

Para llegar a desarrollar este proyecto hemos hecho acopio de información de los productos existentes en el mercado, en cuanto a componentes, sensores y demás elementos relacionados con la robótica se refiere, con la finalidad de crear un robot clónico (creado con diferentes componentes del mercado) capaz de alcanzar los objetivos marcados.

Para ello debíamos ser capaces de ingeniar como conseguir, o mejor dicho, de que manera nuestro robot iba a ser capaz de recorrer un espacio sin chocar con los posibles elementos que se encontrara, de forma autónoma y que además permitiera de algún modo, en momentos de necesidad, ser comandado para poder llegar a cualquier lugar y recabar información.

Una vez determinado el "como", debíamos ser capaces de enlazar todos estos elementos y hacer que interactuaran entre sí, para finalmente ser dirigidos por un cerebro autónomo o unidad de control, que permitiera alcanzar nuestros objetivos de manera eficaz. En este punto es realmente donde se comienza a complicar nuestro camino, puesto que debíamos por un lado ser capaces de elegir correctamente todos los elementos que iban a componer nuestro robot y además ser capaces de programarlos para crear un software que sería el encargado de coordinar todos los movimientos de nuestro robot en función de las diferentes situaciones con las que pudiera encontrarse.

Para llegar a este punto debíamos también diseñar un chasis para poder integrar todos los elementos en una plataforma móvil, capaz de desplazarse por todo el entorno con soltura teniendo en cuenta el peso que debía soportar. También debíamos pensar como alimentar el robot, con que tipo de batería de las existentes en el mercado podíamos alimentar nuestro proyecto para un buen funcionamiento del mismo sin incrementar demasiado el peso.

En conclusión podíamos dividir la ejecución o realización de nuestro proyecto en dos fases:

Diseño y ensamblaje:

-Estudio y elección de los componentes de nuestro robot.
-Ensamblaje y conexionado de los componentes.
-Pruebas de funcionamiento de todos los componentes una vez unificados.
-Resolución de los posibles inconvenientes encontrados después del ensamblaje.

Creación del Software de programación:

-Hemos diseñado una serie de programas básicos para comprobar únicamente el correcto funcionamiento del diseño realizado.

Estas dos fases de realización del proyecto debían ir unidas, ya que es necesario conocer que limitaciones aportan los componentes a través de pequeños programas de ejemplo para conocer mejor el funcionamiento de los mismos.

Una vez llegados a estas conclusiones, decidimos realizar tres pequeños proyectos separados. Cada uno de estos sería un robot que realice una función distinta, a modo de ejemplos que refuercen las aportaciones de este proyecto para la realización de una base de información útil que sea de fácil acceso para futuros proyectos e investigaciones.

2.5 Descripción General:

En esta universidad se desea abrir una nueva línea de investigación sobre robótica, enfocada hacia la competición en concursos como pueden ser el Eurobot (a nivel europeo) y CEABOT, Malakabot, Concentración CAMUBI, CampusBot, Robolid y Semana de la Robótica Alcabot-Hispabot (a nivel español).

Con esa idea, nos ha parecido un buen camino ha seguir el de realizar un proyecto que se centre en explicar como utilizar una serie de componentes que se pueden ser útiles para el desarrollo de un robot para este tipo de competiciones.

En un principio se podría utilizar cualquier placa controladora, pero la universidad nos ha proporcionado un material específico del entorno ARDUINO. Tras un análisis del material proporcionado y después de estudiar otras posibilidades decidimos trabajar con las placas controladoras de Arduino Duemilanove con un AVR ATmega328 y Arduino Mega con un AVR ATMEGA1280; además de con los componentes de este entorno.

Tras una fase de estudio hemos podido comprobar como la placa que tenemos es una herramienta muy útil y potente para este tipo de proyectos, ya que además de sus buenas prestaciones tiene toda una serie de componentes asociados que son de gran utilidad, así como motores, sensores, ruedas, kits complementarios y montón de “gadgets” de todo tipo.

También hemos podido encontrar muchísima información sobre el uso de Arduino así como todo tipo de proyectos de que utilizan estas placas controladoras, lo cual ha sido muy útil para comprender su funcionamiento. Además tiene una página (“www.**arduino**.cc/” en ingles, y “www.**arduino**.cc/es/” en su versión en castellano) en la cual nos proporciona mucha información y el software necesario para el manejo del mismo a través del ordenador, para su programación y la extracción de datos. Esta placa es de software libre, lo cual nos da mas razones para confiar en ella para que sea una buena herramienta de aprendizaje.

Nosotros proporcionamos un **MANUAL con el Lenguaje de Programación del entorno Arduino**, así como una gran cantidad de pequeños ejemplos explicativos de cada una de las estructuras que se pueden utilizar. Se pueden encontrar también dentro del DVD complementario el software necesario para su utilización con u breve documento explicativo para su instalación.

3 Diseños.

3.1 Metodología utilizada.

Para iniciar el diseño de los BOTS era fundamental conocer las funciones que éstos deberían desempeñar y a partir de ese punto conocer todos los dispositivos que se requerirían para llevar a cabo dichas funciones.

Una vez se tubo conocimiento de los dispositivos que se deberían utilizar tales como el sensor seguidor de línea, los servos, las ruedas, los motores, la unidad de control, los pulsadores…etc. nos sumergimos en un gran abanico de posibilidades en cuanto a la elección de dichos componentes, puesto que debíamos encontrar los más adecuados, según sus características, para poder desarrollar la función a la que iban a ser designados de la mejor forma posible y adecuándolos todos ellos a un ensamblaje de unas dimensiones considerables para poder, cuanto menos, reducir el coste del proyecto pero sin restarle las funciones para las cuales iban a ser diseñados.

Me gustaría destacar la dificultad del apartado de diseño ya que se llevo, junto con la parte de programación y desarrollo de la interface del robot, la mayor parte del tiempo designado para la realización del proyecto.

Tras haber elegido la mayoría de los componentes procedimos al estudio de sus características mecánicas de ensamblaje para poder meternos en el diseño del robot que iba a ser el encargado de contener todos los elementos necesarios para el buen funcionamiento.

Lo primero que debíamos pensar era el material del que se iba a fabricar puesto que pasaría a ser el esqueleto del robot, además debíamos pensar en la forma más adecuada del mismo, en base a poder realizar su función principal y que su fabricación no fuera excesivamente complicada y por lo tanto costosa también a nivel económico. Para ello se contemplaron infinidad de modelos ya existentes de todas las vertientes robóticas de la actualidad, con el fin de poder llegar a inspirarse en un diseño práctico, cuidado estéticamente en la medida de nuestras posibilidades, funcional y de un coste lo más asequible posible. Teniendo en cuenta todos estos requisitos optamos por la fabricación de un chasis de chapa metálica, ya que esta elección nos facilitaba muchísimo tanto la elaboración del mismo como la posibilidad de poder realizar pequeñas modificaciones una vez fabricado, debido principalmente, a la maleabilidad y robustez que nos ofrecía.

Una vez realizados unos cuantos bocetos nos dispusimos a establecer de forma aproximada las dimensiones de nuestro robot en base a las necesidades de movilidad que éste debería ofrecernos y a los componentes seleccionados para desarrollar sus funciones.

3.2 Recursos utilizados

Arduino:

Arduino es un entorno, libre de licencias, que nos ha permitido realizar toda la programación de la placa de control de nuestro robot.

Este entorno utiliza, como lenguaje de programación, un lenguaje similar al C que es un lenguaje bastante conocido a nivel de programación que nos ha facilitado, una vez realizado el apartado de diseño, el poder ponernos a trabajar con este entorno sin la necesidad de estudiar demasiado tutoriales para desarrollar nuestro proyecto, puesto que ya teníamos asumidos, a nivel conceptual, los conocimientos necesarios para programar en dicho lenguaje.

A pesar de ello si hemos tenido que conocer el funcionamiento a nivel de programación de todos los sensores con los que cuentan nuestros robots, que ha sido un trabajo un tanto complicado pero que ha dado sus frutos y por lo tanto se ha convertido en un trabajo satisfactorio y enriquecedor, todo ello a pesar de que en realidad nosotros hemos realizado el diseño de una programación mínima de funcionamiento.

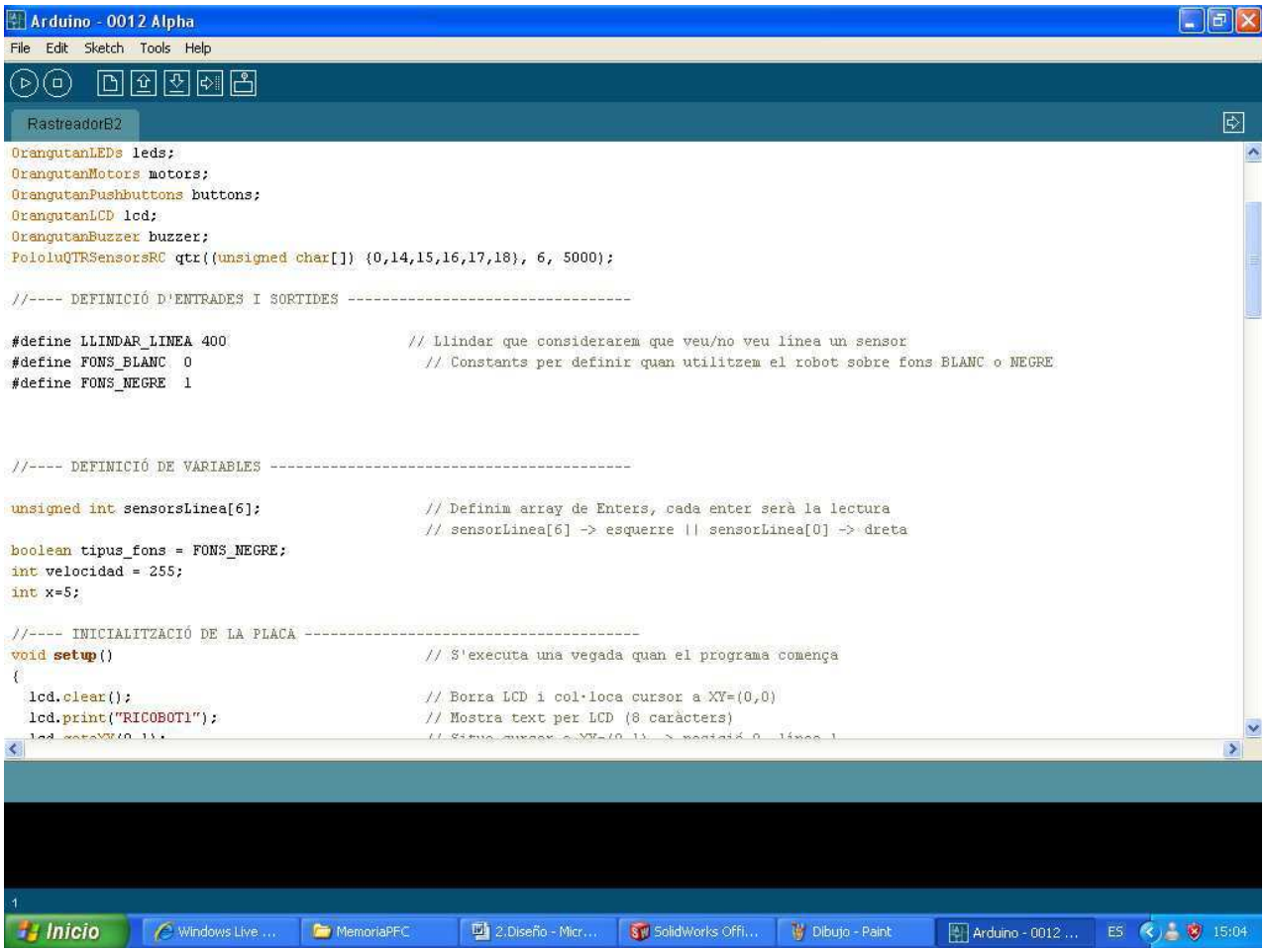


Figura 1. Entorno de Arduino.

Herramientas utilizadas y estación de trabajo:

Además de todos los recursos informáticos, también hemos utilizado toda una serie de herramientas para la realización del ensamblaje de nuestro robot.
Concretamente hemos utilizado como espació físico para nuestro propósito el Laboratorio de Energías Renovables del Edificio de los Pinos de la Universidad Pública de Navarra, donde hemos utilizado un banco de trabajo totalmente acondicionado para ello con herramientas de todo tipo.

Las herramientas que más hemos utilizado de este banco de trabajo son:

-Soldador: para todas las tareas de soldadura de cables, pins de conexionado y calentamiento de las fundas retractiles designadas para aislar y proteger los elementos soldados.
-Taladro: para la realización de los orificios tanto en el chasis como en los herrajes de los servos diseñados por nosotros.
-Destornilladores: Para el ensamblaje de todos los elementos en el chasis y en las salidas PWM de la placa de control.
-Pelacables: para facilitar la tarea de pelar los terminales de los cables.
-Tornillo de banco: Para la sujeción de piezas en su mecanizado además de la realización de dobleces en los herrajes de los servos.
-Tester: Para la comprobación del conexionado y verificación tanto de la polaridad como de la continuidad de las líneas soldadas.
-Martillo: Para realizar dobleces de chapas.
-Reglas: Para tomar medidas a todos los elementos del robot y posteriormente reproducirlos.

Por otro lado también tuvimos que realizar varios cables con distintos conectores. Ni que decir tiene que hemos utilizado también diversos ordenadores como herramientas informáticas para las diferentes tareas comentadas referidas al uso de esta herramienta.

3.3 Descripción del diseño preliminar.

Inicialmente, cuando nos propusimos la realización de este proyecto y nos introducimos en el apartado de diseño contemplamos una serie de componentes un tanto diferentes los cuales finalmente hemos utilizado en el diseño final. Comenzamos realizando unos diseños de las bases de los trés robots en madera, ya que es un material de facil manejo, obteniendo tres diseños preliminares en madera. En el diseño definitivo hemos mantenido el esquema prácticamente igual.

3.4 Descripción del diseño definitivo.

Finalmente una vez finalizados los diseños preliminares nos dispusimos a la realización al completo del diseño definitivo. Para ello contábamos con la dificultad añadida de encontrar componentes compatibles con nuestra placa de control además de a posteriori, una vez seleccionado dichos componentes, entender el funcionamiento de estos y ser capaces de hacerlos funcionar correctamente, configurarlos y programarlos para ejecutar el cometido para el que habían sido seleccionados.

Ultimados estos detalles y los derivados de la adaptación de nuestro chasis a la configuración final con una locomoción que iba a estar formada por dos ruedas motrices de y por dos motores d.c. con reductora 30:1, y una rueda loca en la parte delantera para equilibrar, tenemos por fin la composición completa de elementos que forman nuestro robots y que aunque ahora se enumeren de una forma breve, veremos una descripción mas exhaustiva de cada uno de los componentes en el siguiente apartado.

- Chasis: es el soporte de los componentes y a su vez es el que nos determinará la forma de nuestro robot.

- Placa de control: Es el cerebro del robot se encarga de administrar la información recogida por los sensores y ejecutar las acciones para las que ha sido programada inicialmente por nosotros.

- Motor Shield control v3.0: Controlador de motores específicamente diseñado para placas de control de Arduino. Contiene un puente H imprescindible para el control de motores.

- Motores DC: Se encargan de poner en movimiento las ruedas de nuestro robot para proporcionar al mismo de un medio de locomoción.

- Servos: Con estos componentes realizaremos el movimiento de los sensores con giros de izquierda-derecha o arriba-abajo para captar todo el entorno del robot.

- Batería Li-Po: Es la encargada de administrar la energía eléctrica a todos los componentes de nuestro robot.

- Sonar (sensor de ultrasonidos): Este dispositivo nos permitirá localizar los objetos del entorno para poder ser esquivados por el robot.

- Sensor seguidor de línea (sensores de infrarrojos): Con este elemento nuestro robot será capaz de detectar un recorrido marcado en el suelo y seguirlo sin salirse de éste de forma automática.

- Pantalla LCD: Es el dispositivo encargado de aportar información del robot en tiempo real.

- Teclado numérico: Con este teclado somos capaces de hacer un interfaz para indicar los movimientos que queremos desarrollar.

Esta es principalmente la descripción de nuestro diseño aunque no se han nombrado elementos necesarios para la conexión de los componentes tales como, cableado, pulsadores y conectores puesto que consideramos que son elementos secundarios.

3.5 Descripción de los componentes del diseño.

3.5.1 Chasis

El chasis es una parte importante del robot ya que debemos recordar que es el esqueleto del mismo. Por ello debemos de tener en cuenta una serie de características que debe cumplir a la hora de realizar la elección del material con el que será confeccionado:

-Debe ser lo suficientemente robusto como para contener el peso de todos los componentes, sin sufrir ningún esfuerzo excesivo en cuanto a fuerzas aplicadas se refiere.

-Debe ser capaz de soportar pequeños impactos producidos por posibles colisiones en sus desplazamientos, sin sufrir deformaciones.

-Debe ser lo suficientemente maleable para facilitarnos el proceso de mecanizado.

-Debe ser lo suficientemente liviano como para poder ser arrastrado por los componentes motrices del robot (motores, ruedas) sin que estos sufran un esfuerzo excesivo y por lo tanto un desgaste energético mayor.

-Debe estar realizado con un material que no requiera excesiva dificultad para su producción y por lo tanto no nos provoque un coste económico demasiado elevado.

Tras analizar debidamente esta serie de directrices principales que nos definen los objetivos que deberá cumplir el chasis, elegimos inicialmente la chapa de acero inoxidable como material idóneo para la elaboración del mismo, que a pesar de ser más cara en relación a la chapa de hierro común (opción contemplada a posteriori), nos beneficiaba en cuanto a mantenimiento se refiere, ya que no necesitaba una capa de pintura cada cierto tiempo, a pesar de sufrir ambientes más húmedos, puesto que no se oxida y además teniendo en cuenta el espesor de la chapa (1mm), dicho material nos ofrecía ligeramente una mayor resistencia. Pero determinamos finalmente que como nos encontrábamos en la realización de un prototipo, podíamos realizarla en chapa de aluminio común ya que nos servía de igual forma para los propósitos contemplados anteriormente y además nos representaba un coste económico menor.

Una vez conocido el material y las características que debe cumplir, debemos decir que el chasis en cuestión, esta constituido por dos partes totalmente diferenciadas y con diferentes funciones asignadas cada una. La base a la que estará adosada la placa controladora así como los motores, sensores y demás componentes, y una carcasa exterior para la protección de esos componentes. Ambas partes constituyentes tienen como objetivo común contener los diferentes elementos funcionales del robot.

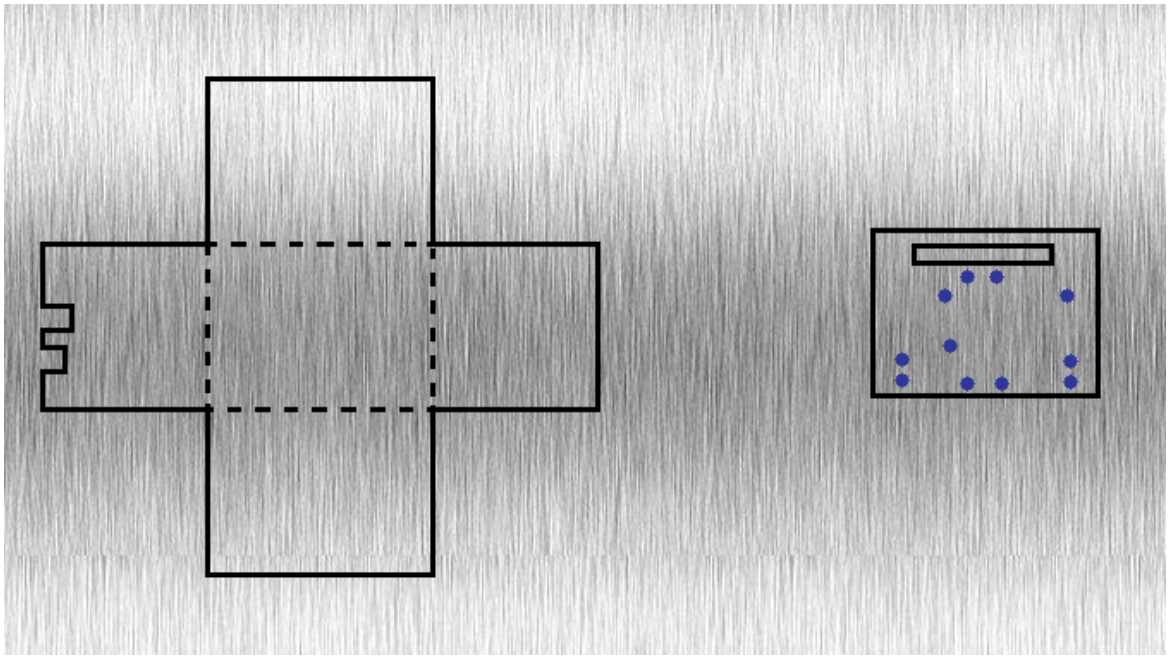


Figura 2. Estructura seguidor de línea impresa sobre una plancha de aluminio

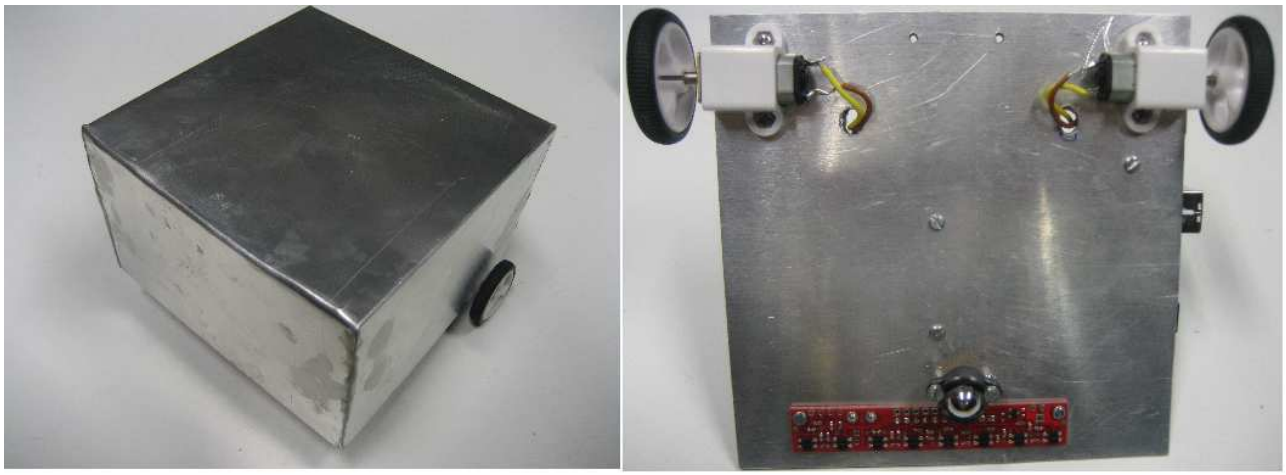


Figura 3. Chasis estructural con componentes y carcasa exterior, del seguidor de línea UPNA

- Chasis estructural: esta parte del chasis además se encarga de contener la batería de Litio-Polímero y de reforzar estructuralmente el chasis además de contener nuestros componentes electrónicos.
- Carcasa exterior: esta parte realiza íntegramente la función descrita anteriormente.

3.5.2 Placa de control

El cerebro del robot. Esta parte es la más importante de nuestro robot, puesto que es la encargada de interpretar y administrar toda la información recibida por los sensores instalados y además es capaz a posteriori de ejecutar las órdenes para las que ha sido programada de una forma rápida y eficiente.
A la hora de su elección nos hemos basado en dos condiciones que eran fundamentales para el desarrollo del robot, una era el lenguaje de programación que utilizaría, que preferíamos que fuera un lenguaje maquina estandarizado en el mundo de la programación, y que por lo tanto no nos dificultara excesivamente la creación y el diseño de los programas en los que se iba a basar el robot, para la correcta finalización de las funciones designadas.
Por otra parte debía tener el número suficiente de entradas y salidas, tanto analógicas como digitales para poder comandar todos los sensores y componentes motrices con los que cuenta el robot.
Una vez analizados detenidamente todos estos detalles y observando el material que disponíamos comenzamos a estudiar las placas controladoras de Arduino que utilizan un entorno de programación que esta libre de licencias, y por lo tanto es accesible y gratuito para todo el mundo, y además dicho entorno (ARDUINO) utiliza un lenguaje parecido al C, para la programación del mismo, que es un lenguaje conocido en el mundo de la programación y que nos ha permitido alcanzar la totalidad de los propósitos de este proyecto.
Por otro lado esta unidad de control, cuyo microcontrolador es el ATMEGA168, encajaba perfectamente, a nivel de comunicación y ensamblaje con los sensores y componentes que estimábamos oportunos para el buen funcionamiento de nuestro robot.

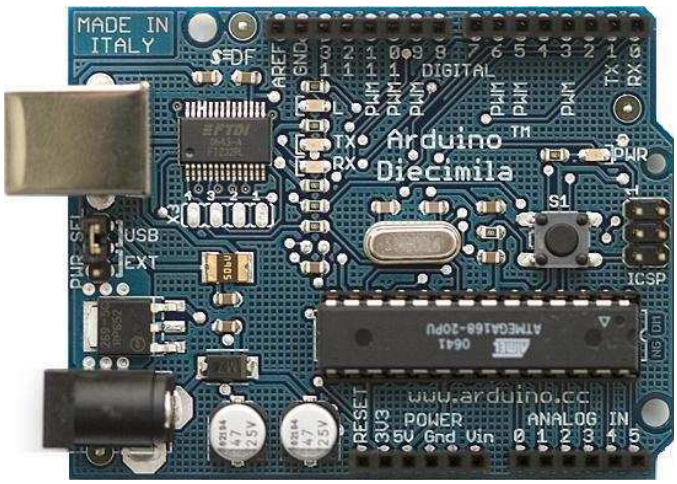


Figura 4. Placa Arduino Duemilanove con un AVR ATmega328

3.5.3 Motores

3.5.3.1 Motores DC

Tras analizar el peso de todo el conjunto y determinar el voltaje de entrada de los motores decidimos montar motores con una reducción de 30:1. Estos motores de corriente continua permiten una fuerza de torsión de 0.4 Kg./cm.

DC Micro Metal de corriente continua con reductora diseñado y fabricado por Pololu para uso en robótica. Nuestros Motores para Robótica son de alta calidad, de dimensiones reducidas y con una reductora metálica que reduce las revoluciones y aumenta la fuerza (troqué). Es similar a los modelos de Sanyo.

Especificaciones (Para 6V)**:**

- **Dimensiones:** 24 x 10 x 12 mm
- **Ratio de la reductora:** 30:1
- **Diámetro del eje:** 3,9mm (con ranura de bloqueo)
- **Voltaje nominal:** 6Vcc (puede funcionar entre 3 a 9Vcc)
- **Velocidad de giro sin carga:** 440rpm
- **Consumo sin carga:** 40mA (Máx.: 360mA)
- **Peso:** 10 gramos
- **Troqué:** 0,4 kg-cm (máx.)

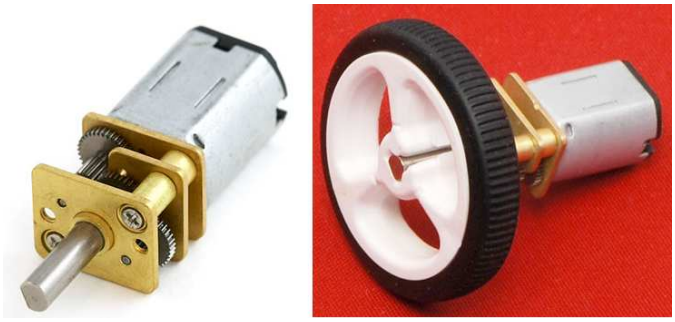


Figura 5. Motores DC Micro metal 30:1

Aunque el motor soporte un amplio rango de voltajes, se recomienda utilizarlo con 6V ya que con voltajes menores pierdes bastante fuerza y con voltajes mayores puede afectar a la vida útil del motor.

Dado su consumo mínimo y reducido tamaño, es ideal para utilizar con integrados de potencia como el ULN2803 o la placa Motor Kit de Arduino.

Para su montaje, se recomienda una funda protectora disponible opcionalmente, la cual incorpora los soportes y tortillería.

El resto de detalles técnicos los podemos encontrar en el datasheet de los mismos situado en el anexo D de este proyecto.

3.5.3.2 Servos

Definición y Estructura

Un servomotor (o servo) es un motor de corriente continua que tiene la capacidad de ser controlado en posición. Es capaz de ubicarse en cualquier posición dentro de un rango de operación (generalmente de 180º) y mantenerse estable en dicha posición. Los servos se suelen utilizar en robótica, automática y modelismo (vehículos por radio-control, RC) debido a su gran precisión en el posicionamiento.

En general, los servos suelen estar compuestos por 4 elementos fundamentales:

-Motor de corriente continua (DC): Es el elemento que le brinda movilidad al servo. Cuando se aplica un potencial a sus dos terminales, este motor gira en un sentido a su velocidad máxima. Si el voltaje aplicado sus dos terminales es inverso, el sentido de giro también se invierte.

-Engranajes reductores: Tren de engranajes que se encarga de reducir la alta velocidad de giro del motor para acrecentar su capacidad de torque (o par-motor).

-Sensor de desplazamiento: Suele ser un potenciómetro colocado en el eje de salida del servo que se utiliza para conocer la posición angular del motor.

-Circuito de control: Es una placa electrónica que implementa una estrategia de control de la posición por realimentación. Para ello, este circuito compara la señal de entrada de referencia (posición deseada) con la posición actual medida por el potenciómetro. La diferencia entre la posición actual y la deseada es amplificada y utilizada para mover el motor en la dirección necesaria para reducir el error.

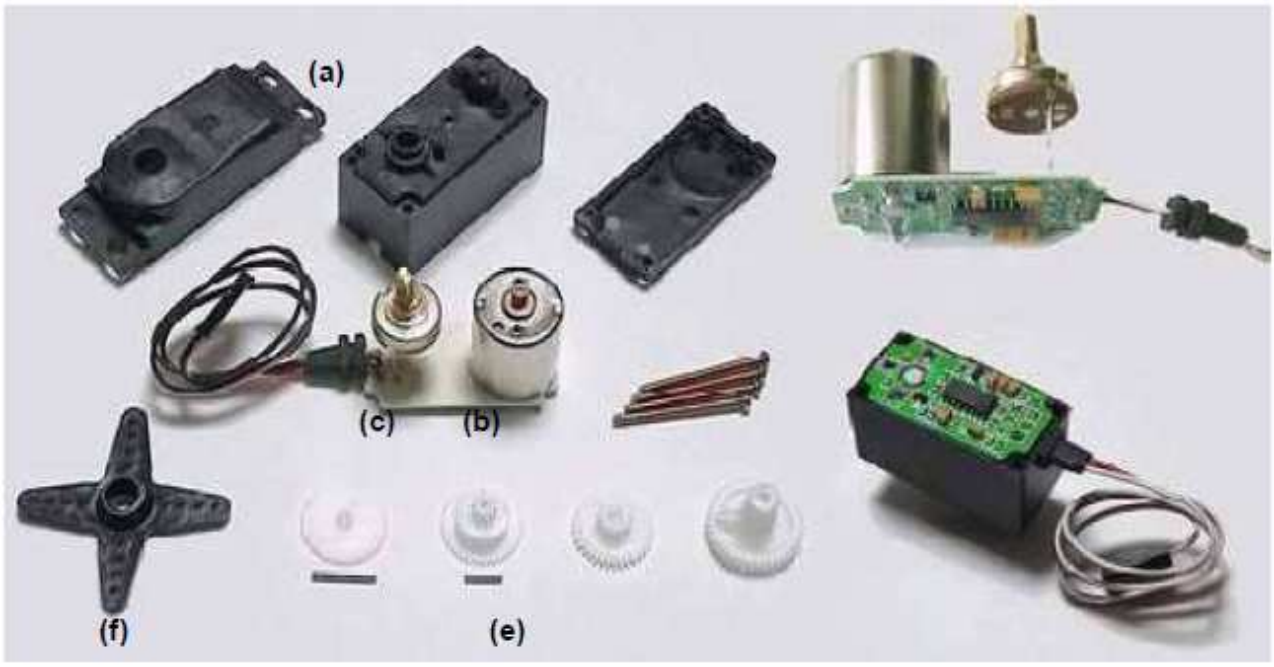


Figura 6. Componentes de un servo: a) carcasa; b) motor DC; c) potenciómetro; d) circuito de control; e) tren reductor; f) brazo (elemento terminal en el eje). Fuente: [1].

Principios de Funcionamiento

Los servos disponen de tres cables (Figura 7): dos cables de alimentación (positivo y negativo/masa) que suministran un voltaje 4.8-6V y un cable de control que indica la posición deseada al circuito de control mediante señales PWM ("Pulse Width Modulation").

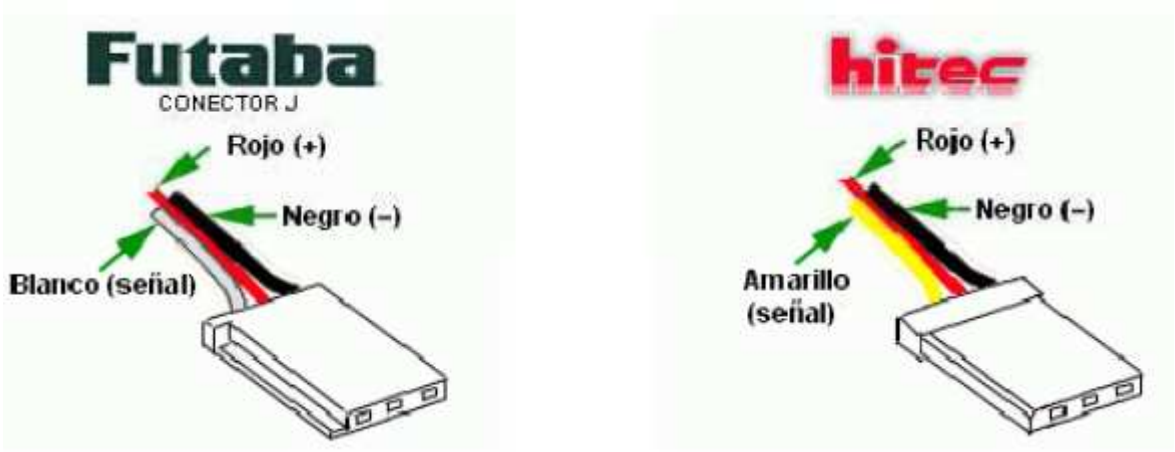


Figura 7. Colores de los cables de los principales fabricantes de servos. Fuente: [1].

Las señales PWM utilizadas para controlar los servos están formadas por pulsos positivos cuya duración es proporcional a la posición deseada del servo y que se repiten cada 20ms (50Hz).
Todos los servos pueden funcionar correctamente en un rango de movimiento de 90º, que se corresponde con pulsos PWM comprendidos entre 0.9 y 2.1ms. Sin embargo, también existen servos que se pueden mover en un rango extendido de 180º y sus pulsos de control varían entre 0.5 y 2.5ms (Figura 8). Antes de utilizar un servo habrá que comprobar experimentalmente su rango de movimiento para no dañarlo. Para mantener fijo un servo en una posición

habrá que enviar periódicamente el pulso correspondiente; ya que si no recibe señales, el eje del servo quedará libre y se podrá mover ejerciendo una leve presión.

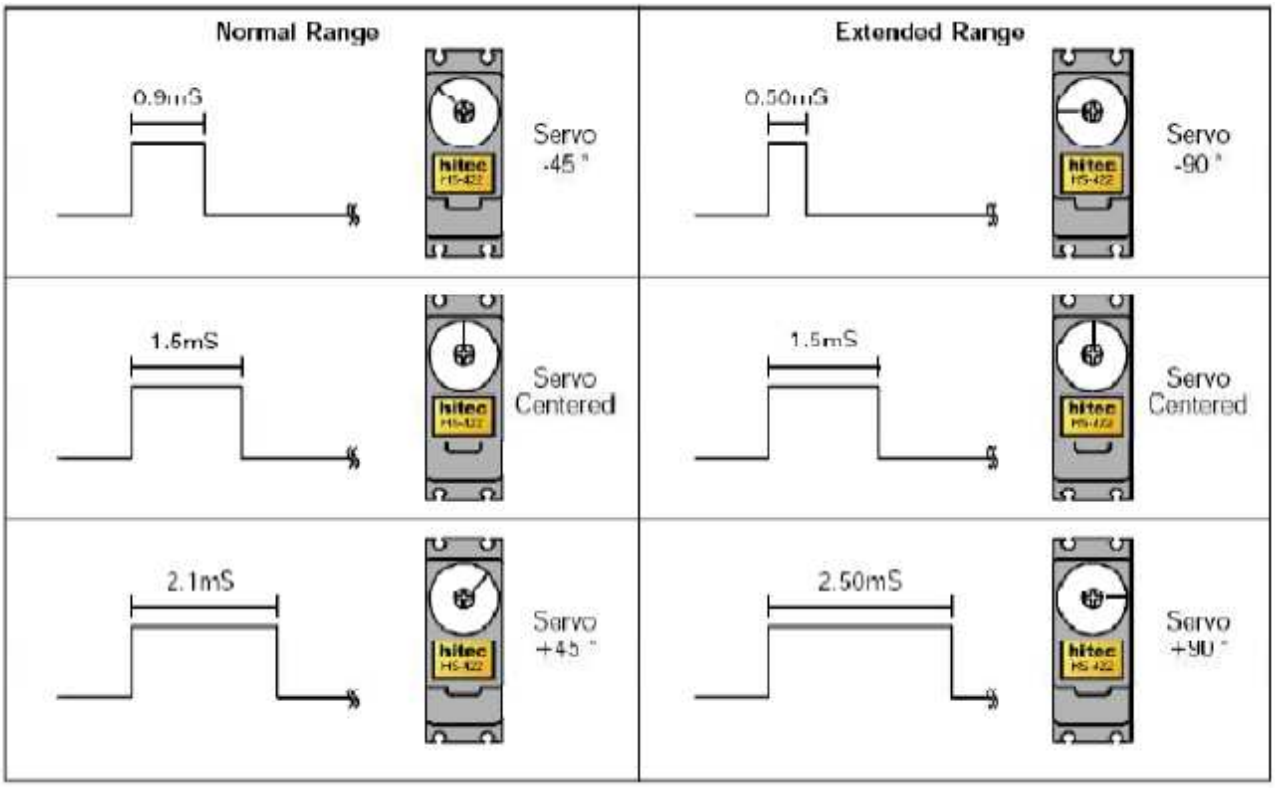


Figura 8. Pulsos PWM para controlar servos. Fuente: [2].

Tipologías

Existen dos tipos de servos: analógicos y digitales. Ambos tipos de servos son iguales a nivel de usuario: tienen la misma estructura (motor DC, engranajes reductores, potenciómetro y placa de control) y se controlan con las mismas señales PWM. La principal diferencia entre ellos radica en la adición de un microprocesador en el circuito de control de los servos digitales [3]. Este microprocesador se encarga de procesar la señal PWM de entrada y de controlar el motor mediante pulsos con una frecuencia 10 veces superior a los servos analógicos.

El aumento en la frecuencia de excitación del motor en los servos digitales permite disminuir su tiempo de respuesta (menor deadband), aumentar su resolución de movimiento y suavizar su aceleración/deceleración. El uso de un microprocesador permite también a los servos digitales programar distintos parámetros de configuración que son fijos en los analógicos: sentido de giro, posición central inicial, topes en el recorrido del servo, velocidad de respuesta del servo y resolución. Para establecer estos parámetros se deben utilizar aparatos específicos de cada marca. El principal inconveniente de los servos digitales es que consumen más energía que los analógicos al tener que generar más pulsos de control para el motor.

3.5.4 Sonar (Sensor de ultrasonidos)

El sensor de ultrasonidos utilizado en el robot, Figura 9 es el modelo **PING)))™ Ultrasonic Distance Sensor (#28015)** de Parallax. El sensor es un completo modulo electrónico que incluye su propio procesador para hacer el cálculo de la distancia, junto con el emisor y el receptor de ultrasonidos.

Su funcionamiento consiste en mandar una serie de impulsos ultrasónicos por el aire y luego medir el tiempo que se tarda en recibir el eco cuando el sonido rebota en algún objeto. Una vez el sonido es recibido por el ultrasonido, el robot tiene programada cierta distancia para evitar el objeto, si la distancia medida por el sensor es menor a la distancia programada el robot debe evitar el obstáculo o bien pararse. Si la distancia medida es superior a la distancia programada el robot puede continuar por el mismo camino.

Este sonar es capaz de detectar objetos de 0 a 3,5m, aproximadamente con una resolución de 2,5 cm. para distancias mayores a 15 cm. En diferencia a otros sonar del mercado este no presenta zona muerta, es decir, puede detectar pequeños objetos incluso cuando estos están en contacto físico con la parte frontal del mismo.

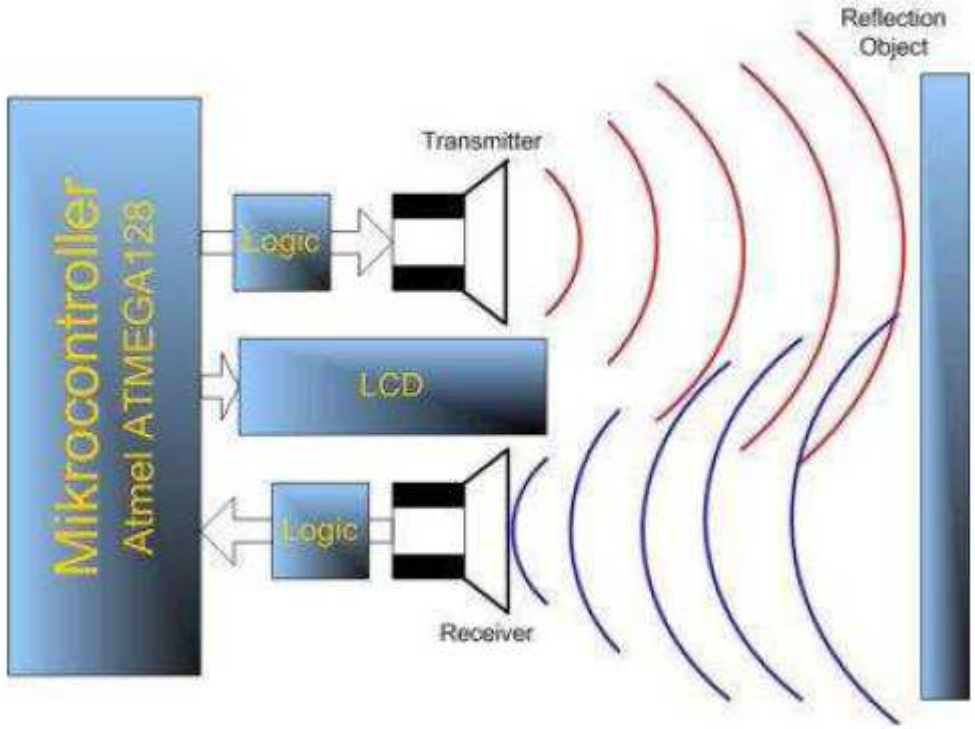


Figura 9. Esquema de funcionamiento.

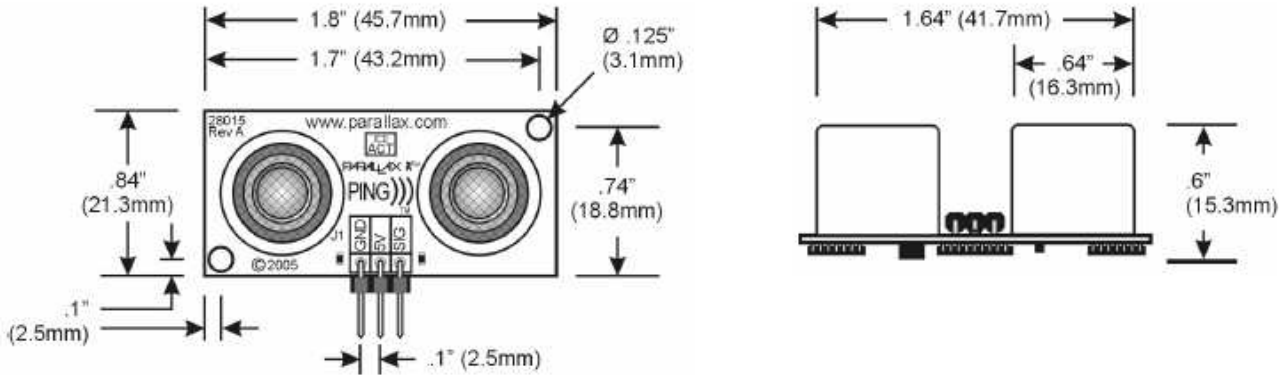


Figura 10. PING))) ™ Ultrasonic Distance Sensor (#28015) de Parallax

3.5.5 Batería LI-PO

La descripción de este componente de nuestro proyecto es algo más extensa debido a la importancia del mismo, puesto que aunque no lo pueda parecer en primera instancia es un elemento que si no se utiliza correctamente puede llegar a ser muy peligroso.

Debido a su importancia hemos querido incluir un manual de instrucciones de utilización de dicha batería en el apartado del anexo A del presente proyecto final de carrera, aunque ahora haremos un breve resumen del funcionamiento y de la elección de dicho componente.

Los beneficios de utilizar esta tecnología son tiempos de funcionamiento más largos, mas potencia, incluso tanta potencia o mas que un motor de combustión interna si el modelo esta bien estudiado.

Por supuesto es necesario aprender a utilizarlas, igual que se ha de aprender a utilizar cualquier otro tipo de baterías. Cuando se aprende, es sencillo mantener todos los elementos en buen estado incluso más simple debido a que no tienen efecto memoria.

En vista de los tiempos de funcionamiento más largos y la relación potencia -peso que se consiguen utilizando Lipos, merece la pena el esfuerzo de aprender a manejarlas correctamente.

Si se saben utilizar y tratar bien a estas baterías, su vida útil es mucho más larga que las baterías de NiCd y NiMh. Pero si no se hace bien se estropean en muy pocos usos (en tan solo 1 incluso) y se pueden convertir en un material realmente peligroso si no se utilizan bien.

Sobre los packs y como elegir el más idóneo:

Primero un poco de teoría sobre la arquitectura de los packs de Lipo:

Un solo elemento de NiCd o NiMh (Nixx) tiene un voltaje de 1,2V, Un elemento de Lipo tiene un voltaje de 3,7V.

Por lo que un elemento de Lipo (3,7V) equivale aproximadamente a 3 elementos de Nixx (3,6V) conectados en serie.

Ahora por ejemplo si se conecta 3 elementos en serie de Lipo de 1500mAh nos dará un voltaje de 11,1V y su capacidad será de 1500mAh y se denominaría 3S (3 en serie) Si a este pack 3S se conecta un pack igual en paralelo tendremos entonces un pack de 11,1V y 3000mAh y se denominaría 3S2P (3 en serie conectado en paralelo a otro de 3 en serie) Si estos dos packs de 3S los conectamos en serie tendremos entonces un pack de 22,2V y 1500mAh y se denominaría 6S (6 en serie) Y de esta manera se pueden hacer los pack como se quiera...

Con los packs de elementos Nixx (Nicd o Nimh) es igual, un pack de 8S2P de elementos NiCd 1700mAh, serian 8 elementos en serie conectado a otro pack en paralelo de 8 en serie...
Quedando como de 9,6V y 3400mAh.

Conclusiones:

· Si se conectan elementos o packs en serie se suman los voltajes. No se deben conectar packs en serie a menos que sean iguales y todos los packs/elementos al conectar deben de estar perfectamente equilibrados entre si. Se pueden cargar estos packs sin desconectar la/s conexión/es en serie.

· Si se conectan elementos o packs en paralelo se suman capacidades tanto de mAh como de capacidad de descarga. Se pueden conectar los elementos/packs en paralelo para conseguir más capacidad y mayor descarga.

Los packs/elementos a conectar en paralelo deben estar ambos al mismo voltaje, normalmente o cargados o descargados. Se pueden cargar a la vez los packs conectados en paralelo si son de la misma capacidad y tipo.
Para elegir el pack más idóneo para nuestro modelo debíamos conocer el consumo máximo real de los motores y demás elementos con su configuración, este consumo máximo debe ser menor que la descarga máxima que el pack es capaz de desarrollar, utilizar un pack de Lipo siempre a su máxima descarga, aunque fuera capaz, no daría el rendimiento deseado y la vida misma del pack se acortaría.

Ejemplo: Para un motor que consume 10A a máximo régimen de vueltas con la rueda montada, deberíamos elegir un pack capaz de dar al menos 15A y recomendado unos 20A.

Para saber el tiempo de funcionamiento que podemos esperar de un pack podemos usar la siguiente formula:

(60 m/h / Consumo del motor en A) x (capacidad de la batería en mAh / 1000m) = Tiempo de funcionamiento en minutos

El tiempo que nos da esta formula seria el tiempo teórico de funcionamiento si estuviéramos utilizando los motores constantemente, lógicamente no siempre se utilizan los motores al máximo en un modelo por lo cual este tiempo de funcionamiento seria mayor.

Además debemos tener en consideración diferentes perdidas en el sistema ya que no se debe vaciar un pack de baterías de Lipo completamente tanto por seguridad del modelo como por seguridad de las baterías, la experiencia nos ha

enseñado que este dato que sale de esta formula es muy aproximado al tiempo de funcionamiento real que podemos esperar del modelo sin entrar en zonas criticas tanto para el modelo como para el pack de baterías.

BATERÍA LIPO 1100MAH *BATERÍA LIPO 2000MAH*

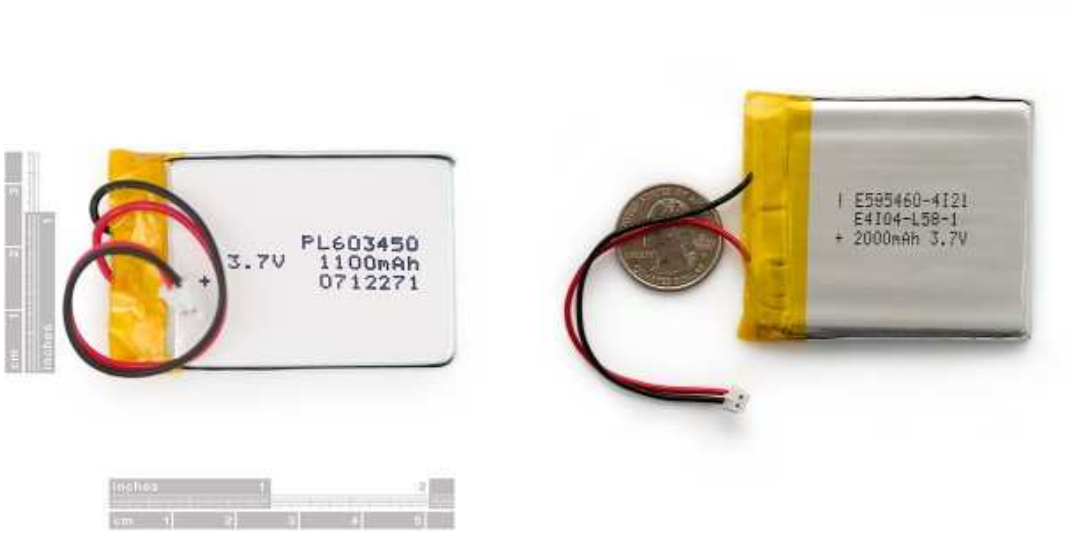


Figura 11. Baterías Li-Po

3.5.6 Sensor Seguidor de Línea.

El sensor seguidor de líneas es un modulo electrónico de Pololu que se coloca en la parte delantera e inferior del robot. Está formado por 8 emisores y detectores de infrarrojos separados 9,5 mm. entre sí de los cuales nosotros hemos utilizado únicamente los cuatro pertenecientes a la parte central.

La función principal del seguidor de líneas es la de detectar mediante los emisores y detectores de infrarrojos si debajo del robot hay algún tipo de línea. Si es así el robot sigue dicha línea intentando no perder su rastro. Si pierde el rastro, el robot debe de intentar volver lo antes posible a la línea para hacer el recorrido.

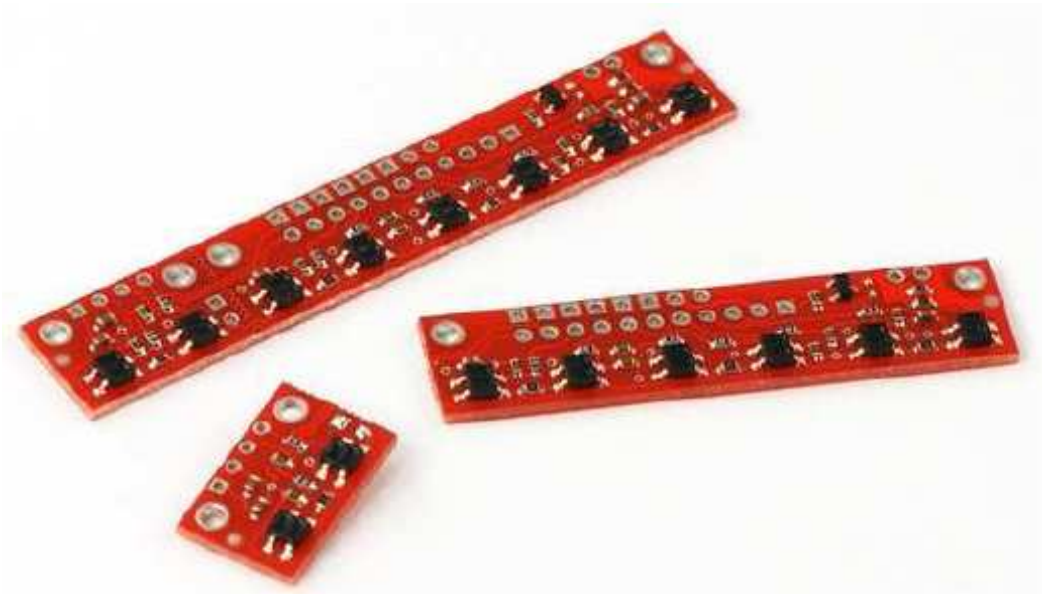


Figura 12. Sensor seguidor de líneas

Teoría del Funcionamiento del sensor seguidor de línea

El circuito funciona de la siguiente manera, el led emisor de los sensores se alimenta a través de una resistencia R1, cuando una superficie reflectante como el color claro de la superficie por donde se moverá el robot en modo rastreador, refleja la luz del led emisor, el fototransistor contenido en el sensor baja su resistencia interna entre Colector y Emisor con lo cual conduce la corriente que hace que también entre en conducción el transistor Q1 que estaba polarizado a masa por medio de la resistencia R2 de 10 K_. Q2 sirve para invertir la señal para que de este modo se desactiven los 5v de la entrada de la señal digital de la placa de control cuando ve blanco y se ponga en marcha cuando ve negro el sensor, con lo que al activarse Q1 hace que se active Q2 cortando a Q3 con el, ya que este ultimo estaba activo porque esta polarizado por R3, con lo cual lo que ha pasado es que la salida de la placa se ha desactivado cuando el sensor ha detectado una superficie reflectante, en estado de reposo la salida estará siempre activa y Q3 conduciendo. Los 2 circuitos se pueden alimentar con 4 pilas normales de 1,5V puestas en serie con lo que se obtienen 6V, dependerá del consumo de los motores elegir pilas o baterías mas potentes.

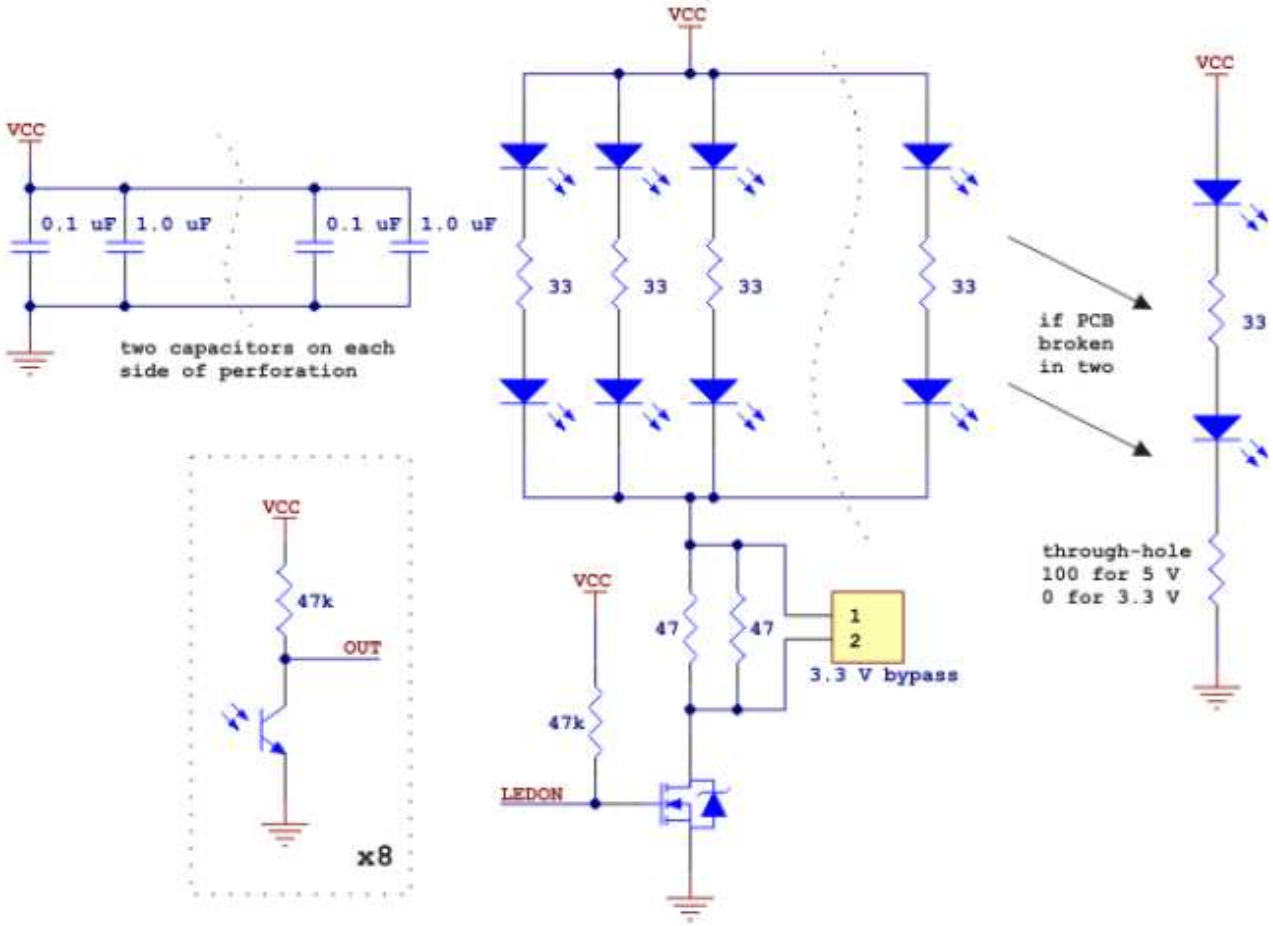


Figura 13. Esquema eléctrico de los led emisores y receptores ubicados en el sensor seguidor de líneas.

3.5.7 Motor_control_v3.0_sch

Es un controlador que se utiliza par que la placa arduino pueda controlar motores del tipo servo, stepper o motores DC. Su principal componente es un puente H, imprescindible para el control de motores.

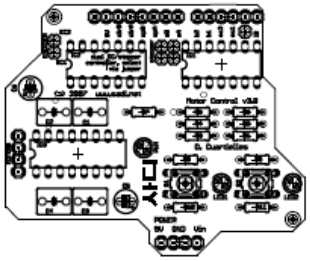
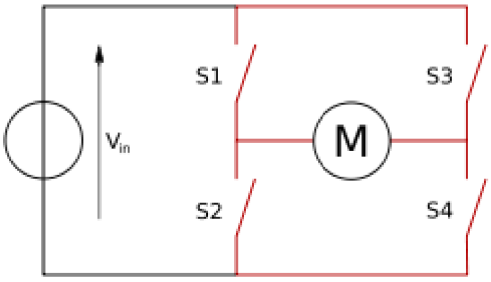
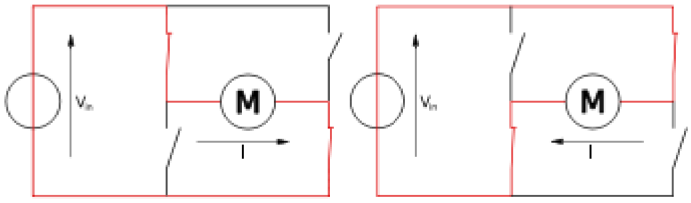


Figura 14. Motor Shield v3.0

Un **Puente H** o **Puente en H** es un circuito electrónico que permite a un motor eléctrico DC girar en ambos sentidos, *avance* y *retroceso*. Son ampliamente usados en robótica y como convertidores de potencia. Los puentes H están disponibles como circuitos integrados, pero también pueden construirse a partir de componentes discretos.



Estructura de un puente H (marcado en rojo).



Los 2 estados básicos del circuito.

El término "puente H" proviene de la típica representación gráfica del circuito. Un puente H se construye con 4 interruptores (mecánicos o mediante transistores). Cuando los interruptores S1 y S4 (ver primera figura) están cerrados (y S2 y S3 abiertos) se aplica una tensión positiva en el motor, haciéndolo girar en un sentido. Abriendo los interruptores S1 y S4 (y cerrando S2 y S3), el voltaje se invierte, permitiendo el giro en sentido inverso del motor.
Con la nomenclatura que estamos usando, los interruptores S1 y S2 nunca podrán estar cerrados al mismo tiempo, porque esto cortocircuitaría la fuente de tensión. Lo mismo sucede con S3 y S4.

Aplicaciones
Como hemos dicho el puente H se usa para invertir el giro de un motor, pero también puede usarse para frenarlo (de manera brusca), al hacer un corto entre los bornes el motor, o incluso puede usarse para permitir que el motor frene bajo su propia inercia, cuando desconectamos el motor de la fuente que lo alimenta. En el siguiente cuadro se resumen las diferentes acciones.

S1	S2	S3	S4	Resultado
1	0	0	1	El motor gira en *avance*
0	1	1	0	El motor gira en *retroceso*
0	0	0	0	El motor se detiene bajo su inercia
0	1	0	1	El motor frena (*fast-stop*)

Montaje
Lo más habitual en este tipo de circuitos es emplear interruptores de estado sólido (como Transistores), puesto que sus tiempos de vida y frecuencias de conmutación son mucho más altas. En convertidores de potencia es impensable usar

interruptores mecánicos, dado su bajo número de conmutaciones de vida útil y las altas frecuencias que se suelen emplear.
Además los interruptores se acompañan de diodos (conectados a ellos en paralelo) que permitan a las corrientes circular en sentido inverso al previsto cada vez que se conmute la tensión, puesto que el motor está compuesto por bobinados que durante breves períodos de tiempo se opondrán a que la corriente varíe.

3.5.8 Pantalla LCD

Norma LCD HD44780 son útiles para la creación de proyectos independientes.

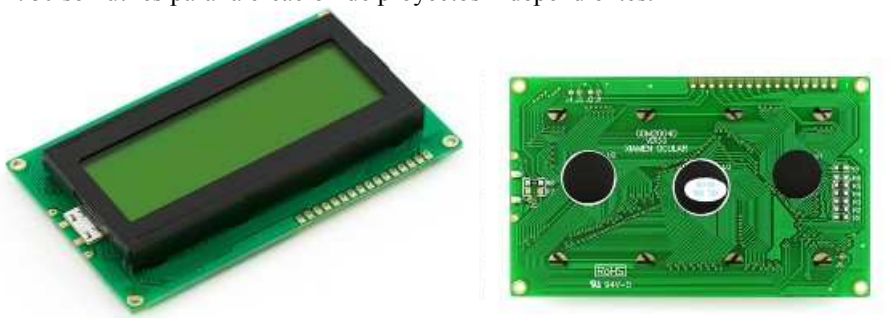


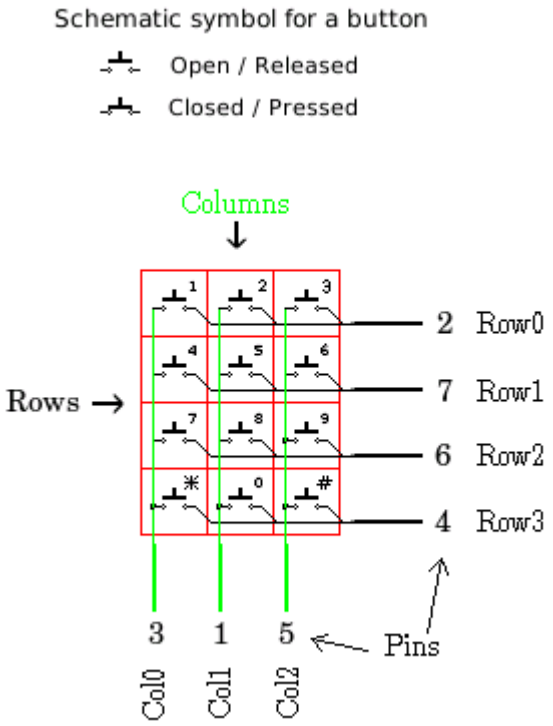
Figura 15. Pantalla LCD de 20x4 caracteres

Características:

- 20 caracteres de ancho, 4 filas.
- Texto negro sobre fondo blanco.
- Retroiluminación LED individuales incluidos se pueden regular con facilidad con una resistencia o PWM.
- Puede ser totalmente controlado con sólo 6 líneas digitales.
- Construido en el conjunto de caracteres compatible con la mayoría Inglés / Europa / texto en japonés, consulte la hoja de datos HD44780 para el carácter del sistema completo.
- Hasta 8 caracteres adicionales se pueden crear para glifos personalizados o "extranjero" soporte de idiomas.
- Viene con potenciómetro de contraste necesario y bandas de cabecera.

3.5.9 Teclado numérico matricial

Teclado matricial con 12 botones montado en matriz, por lo que es ideal para usar con cualquier microcontrolador interpretando los pines para averiguar qué botón se encuentra pulsado.
Millones de posibilidades. Control de accesos, juegos, montajes para la red telefónica, etc.



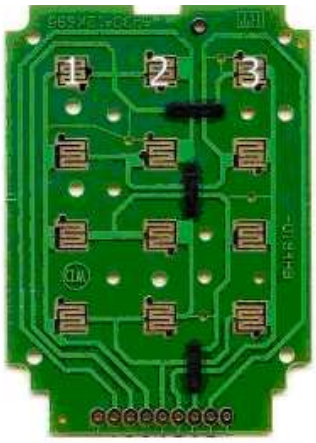


Figura 16. Teclado Numérico matricial y esquema de conexiones internas

4. Desarrollo de los robots

4.1 Diseño e Implementación

En esta parte del proyecto vamos a explicar detalladamente de que manera hemos realizado los tres robots de ejemplo que hemos desarrollado en este PFC. Cada uno de ellos tiene partes en común como pueden ser la parte de movimiento, pero cada uno tiene elementos únicos.

Inicialmente hemos comprobado el buen funcionamiento mecánico de los motores y de la placa de control de los robots así como la correcta realización de los giros en ambas direcciones diseñando un programa básico para este propósito. Estos programas están incluidos en la parte final de este proyecto final de carrera.

4.1.1 Robot 1: "No Me Txoko"

En este caso hemos desarrollado un robot que se mueva por cualquier estancia evitando chocarse con los distintos obstáculos que estén en la misma. La lista de componentes que utiliza es la siguiente:

- Placa de control Arduino Duemilanove
- Motor Shield control v3.0
- 2 Motores DC con sus ruedas
- Rueda loca
- Servo motor
- Sonar (sensor de ultrasonidos) PING)))™ Ultrasonic Distance Sensor (#28015)
- 2 Baterías Li-Po, 1 de 1000mAh y otra de 2000mAh
- Chasis de metal

Movilidad

En este primer robot vamos a explicar como se realiza la parte del movimiento, que será igual para los siguientes robots, así que a fin de no repetir la información tomaremos esto igual para los otros robots.

Comenzamos comprobando el buen funcionamiento mecánico de los motores conectándolos a una corriente, primero en un sentido y a continuación con una corriente contraria en el sentido opuesto.

Ahora es necesario conectar la placa controladora Arduino Duemilanove con la placa de control de motores Motor Shield v3.0. Con esto podemos conseguir que la placa nos proporcione el voltaje adecuado para mover los motores. Es necesario utilizar una placa controladora de motores que contenga un puente H, diseñado para este fin. También podemos realizar esta placa manualmente, aunque recomendamos servirnos de una que ya venga con todo lo necesario. Nosotros adquirimos una Motor Shield v3.0 totalmente desmontada, y le tuvimos que soldar todos los componentes, por lo que hemos añadido un video explicativo que se puede encontrar en la página de Bricogeek y en el DVD complementario adjuntado al PFC.

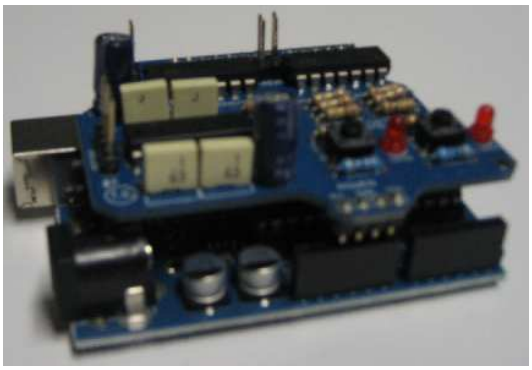


Figura 17. Placa controladora conectada al controlador de motores.

La conexión se debe hacer tal y como aparece en el siguiente esquema de conexionado:

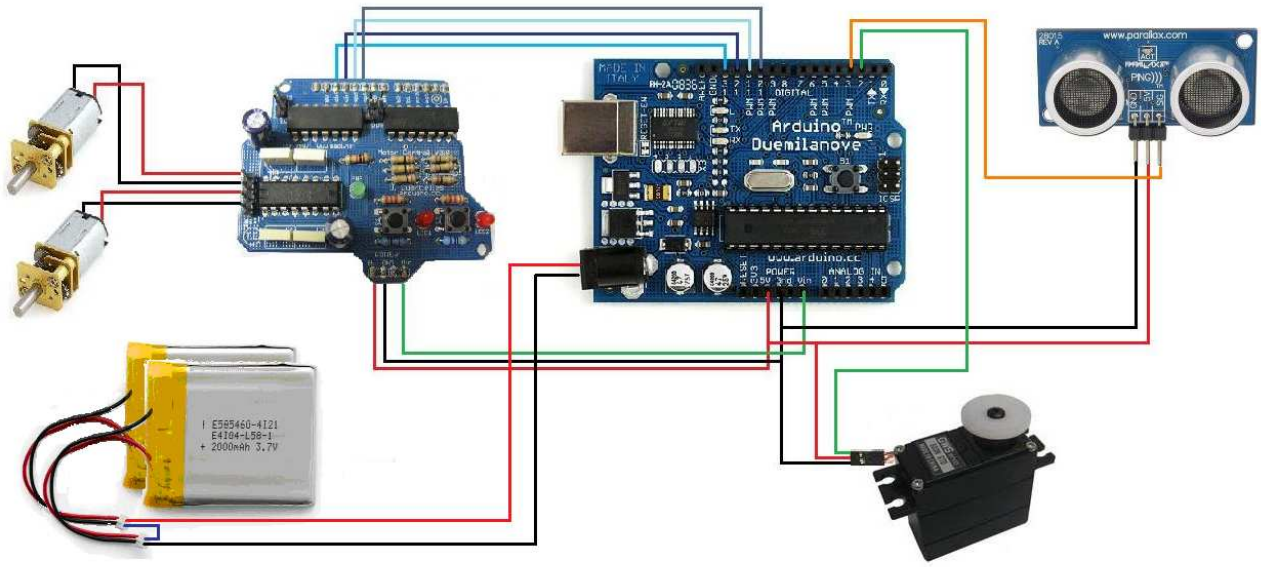


Figura 18. Esquema de conexionado.

Una vez conectada tal y como aparece en la figura conectamos los motores DC en los pines adecuados para estos en la palca controladora de lo motores. Introduciendo la información adecuada en los pines de dirección y velocidad conseguimos que los motores se muevan tal y como queremos.

Para nuestro caso particular tenemos la siguiente conexión:

- Arduino: PIN 10 → Motor Shield: PIN dirA
- Arduino: PIN 9 → Motor Shield: PIN dirB
- Arduino: PIN 11 → Motor Shield: PIN speedA
- Arduino: PIN 12 → Motor Shield: PIN speedB
- Arduino: PIN 3 → Señal Ping del sensor
- Arduino: PIN 2 → PIN de control del Servo motor
- Arduino: PIN Gnd → Motor Shield: PIN Gnd, Servo motor Gnd, y Gnd del sensor de ultrasonidos
- Arduino: PIN Vcc → Motor Shield: PIN Vcc, Servo motor Vcc, y Vcc del sensor de ultrasonidos

Sensor ultrasonidos

A continuación hemos realizado unas pruebas con el sensor de ultrasonidos comprobando que la señal resultante en un osciloscopio cumple con las especificaciones proporcionadas por el fabricante. Una vez comprendido dicho funcionamiento hemos creado un programa el cual nos proporciona la distancia a la que nos situamos del próximo obstáculo. Para ellos lo que hacemos es emitir una señal de alta frecuencia de una duración determinada y esperamos a que dicha señal sea recibida otra vez por el sensor obteniendo el tiempo que ha tardado en ir y en volver la señal. Con ese tiempo lo que hacemos es dividirlo entre 29 ya que la velocidad de la luz es 29µs/cm y posteriormente entre 2 ya la señal realiza dos veces el mismo camino consiguiendo como resultado la distancia en centímetros al próximo objeto.

En esta parte decidimos colocar el sensor de ultrasonidos acoplado al servo motor de manera que pudiese realizar barridos captando la distancia a la que se encuentra para poder decidir a través de la programación en que dirección seguir. Por eso pensamos que debería ir en la parte delantera del robot para que no le molestase la propia estructura del robot. Comprendimos que debía estar perfectamente alineado ya que sino captaría el suelo dándonos lecturas erróneas.

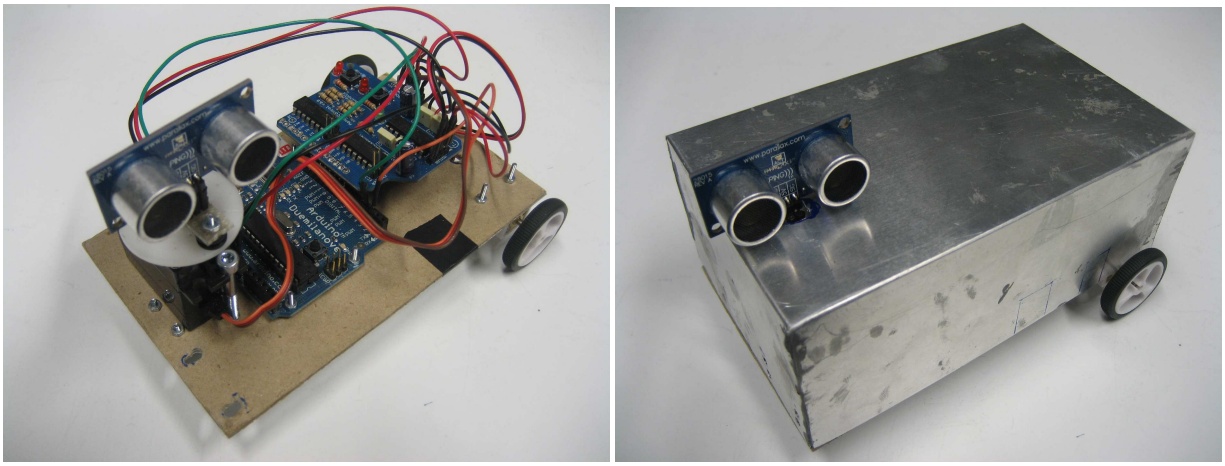


Figura 19. Robot No Me Txoko

Al tener un único sensor hemos optado por una solución en la que realiza una parada cada cierto tiempo para ver que es lo que le rodea y decidir en que dirección continuar, y que mientras tanto realice una lectura continua de lo que tiene enfrente y que si detecta un obstáculo gire y continúe en otra dirección.

Con más sensores se puede conseguir un robot más eficiente que evite mejor los obstáculos y pueda hacerse un mapa de lo que tiene en su entorno de manera continuada, es decir en tiempo real.

Finalmente y tras una serie de pruebas conseguimos afinar el programa para su buen funcionamiento.

Figura 20. Robot No Me Txoko Finalizado

4.1.2 Robot 2: Seguidor de línea

Para este ejemplo se ha realizado un robot que siga una línea negra (o blanca) sobre una superficie blanca (o negra para línea blanca). El diseño que hemos realizado consta de los siguientes componentes:

- Placa de control Arduino Duemilanove
- Motor Shield control v3.0
- 2 Motores DC con sus ruedas
- Rueda loca
- Sensor de infrarrojos QTR-8A Reflectance Sensor Array
- 2 Baterías Li-Po de 2000mAh
- Chasis de metal

Movilidad

Al igual que para el robot anterior hemos utilizado los mismos motores y placa controladora de los mismos, siendo su conexión exacta la siguiente:

- Arduino: PIN 11 → Motor Shield: PIN dirA
- Arduino: PIN 10 → Motor Shield: PIN dirB
- Arduino: PIN 13 → Motor Shield: PIN speedA
- Arduino: PIN 12 → Motor Shield: PIN speedB

- Arduino: PIN 2 → Sensor infrarrojos 1
- Arduino: PIN 3 → Sensor infrarrojos 2
- Arduino: PIN 4 → Sensor infrarrojos 3
- Arduino: PIN 5 → Sensor infrarrojos 4
- Arduino: PIN 6 → Sensor infrarrojos 5
- Arduino: PIN 7 → Sensor infrarrojos 6

- Arduino: PIN Gnd → Motor Shield: PIN Gnd, y Gnd del array de sensores de infrarrojos
- Arduino: PIN Vcc → Motor Shield: PIN Vcc, y Vcc del array de sensores de infrarrojos

Sensor de infrarrojos

El robot cuenta con un programa de uso, en modo autónomo, que recibe una señal de arranque desde la placa de control Arduino Duemilanove con un AVR ATMEGA328 una vez el usuario lo ha conectado a las baterías.

En este modo, tras recibir la señal de arranque, el robot seguirá un recorrido marcado en el suelo que le guiará automáticamente por el tablero en el que se encuentre el robot.

El funcionamiento del robot, desde el punto de vista del usuario, es muy sencillo, ya que éste ha sido diseñado para no complicar ni lo más mínimo el uso del mismo, teniendo en cuenta que ha sido diseñado para que se utilice sin ningún tipo de control externo.

El usuario únicamente tendrá, inicialmente, que situar el robot en la zona deseada, a continuación deberá calibrarlo situando los sensores sobre la parte más oscura (la pista) y la mas clara (el tablero). Esto es necesario para que los sensores se calibren de acuerdo con la iluminación de la estancia.

A partir de este momento el usuario no deberá prestarle ningún tipo de atención más, salvo que hay que recordar que nuestro robot funciona con batería, lo que significa que el usuario deberá de vez en cuando cerciorarse del estado de la misma.

Para comprender mejor el funcionamiento adjuntamos esquemas de la configuración actual y del esquema de conexionado:

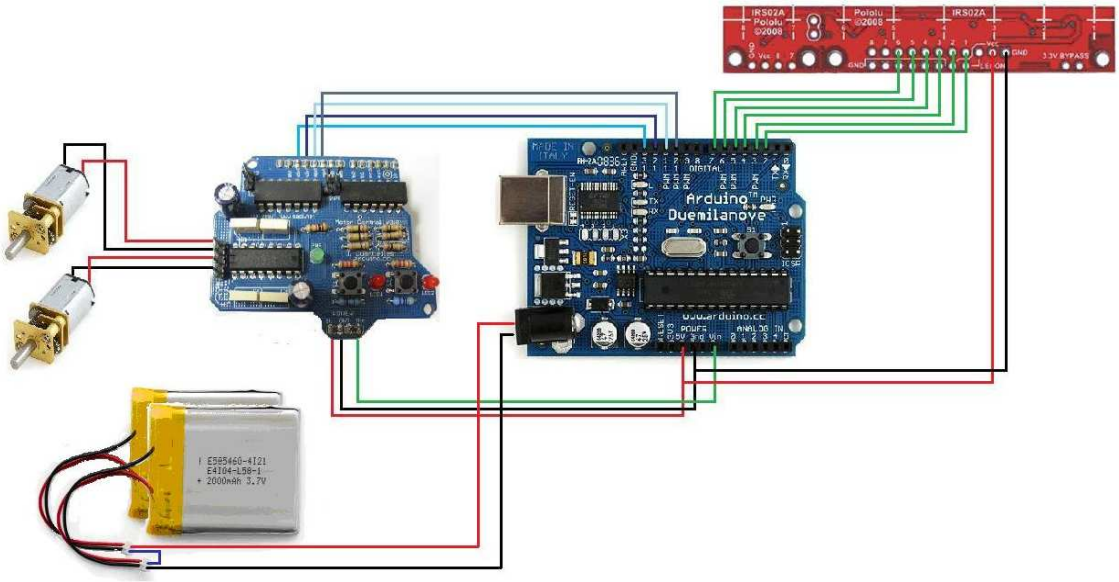


Figura 21. Esquema de conexionado del robot seguidor de línea.

Tras realizar un estudio de cual podría ser el mejor diseño de este robot, y después de varios modelos hemos realizado un el prototipo que a continuación mostramos:

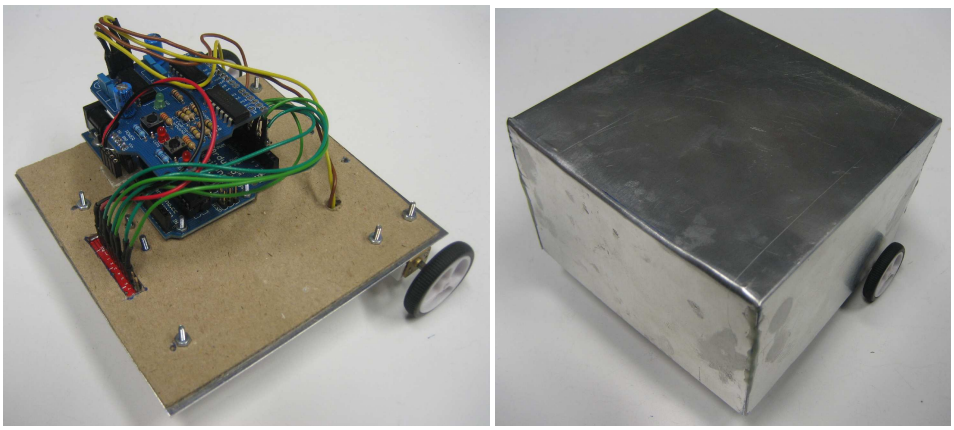


Figura 22. Robot seguidor de línea.

Figura 23. Robot seguidor de línea finalizado.

4.1.3 Robot 3: Bee-Bot

Finalmente hemos construido un robot que realice las mismas funciones que un robot comercial llamado "Bee-Bot" que mediante pulsación y grabación de la misma realiza una serie de pasos previamente programados. La lista de componentes que utiliza nuestro robot es la siguiente:

- Placa de control Arduino Mega
- Motor Shield control v3.0
- 2 Motores DC con sus ruedas
- Rueda loca
- Pantalla LCD
- Teclado numérico
- 2 Baterías Li-Po de 2000mAh
- Chasis de metal

Movilidad

Al igual que para el robot anterior hemos utilizado los mismos motores y placa controladora de los mismos, siendo su conexión exacta la siguiente:

- Arduino: PIN 11 → Motor Shield: PIN dirA
- Arduino: PIN 10 → Motor Shield: PIN dirB
- Arduino: PIN 13 → Motor Shield: PIN speedA
- Arduino: PIN 12 → Motor Shield: PIN speedB

- Arduino: PIN 38 → Sensor infrarrojos 1
- Arduino: PIN 40 → Sensor infrarrojos 2
- Arduino: PIN 42 → Sensor infrarrojos 3
- Arduino: PIN 44 → Sensor infrarrojos 4
- Arduino: PIN 46 → Sensor infrarrojos 5
- Arduino: PIN 48 → Sensor infrarrojos 6

- Arduino: PIN 39 → Columna 1 del teclado numérico
- Arduino: PIN 41 → Fila 0 del teclado numérico
- Arduino: PIN 43 → Columna 0 del teclado numérico
- Arduino: PIN 45 → Fila 3 del teclado numérico
- Arduino: PIN 47 → Columna 2 del teclado numérico
- Arduino: PIN 49 → Fila 2 del teclado numérico
- Arduino: PIN 51 → Fila 1 del teclado numérico

- Arduino: PIN 8 → LED 1 (opcional)
- Arduino: PIN 9 → LED 2 (opcional)

- Arduino: PIN Gnd → Motor Shield: PIN Gnd, y Gnd del array de sensores de infrarrojos
- Arduino: PIN Vcc → Motor Shield: PIN Vcc, y Vcc del array de sensores de infrarrojos

Teclado numérico

Lo que pretendemos es que al pulsar una serie de teclas del teclado numérico el robot se mueva a consecuencia. Queremos que se comporte igual que el robot comercial por lo que queremos que realice cuatro movimientos: adelante, izquierda, derecha y atrás.

La mecánica del programa consiste en recopilar una serie de acciones (p.ej.: adelante, adelante, izquierda, atrás, derecha) y cuando se pulse el botón de comienzo realizar toda la serie hasta el final. Este robot está diseñado para el uso didáctico para niños de corta edad.

La equivalencia entre el teclado y los valores reales de las teclas en el programa es la siguiente:

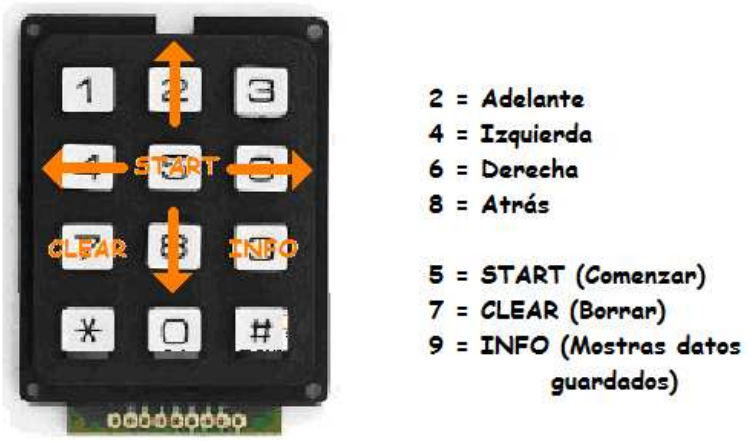


Figura 24. Teclado numérico

Para ver la mecánica con mayo exactitud hemos incluido más adelante el código de los programas que gobiernan estos robots. Mirar el apartado de "Programas de diseño".

Pantalla LCD

Con la pantalla LCD pretendemos mostrar la información del seguimiento del programa. Las acciones que nos va a mostrar son las que puede ir realizando tal y como se ha explicado en el teclado numérico. Así pues cuando vaya a comenzar nos lo indicará en pantalla, y a continuación nos mostrará la acción que está realizando así como la que va a realizar seguidamente; hasta acabar los movimientos e indicarnos que se ha terminado.
En esta pantalla tenemos cuatro filas de 20 caracteres cada una con lo que nos permite también mostrar un vector con la numeración de los movimientos grabados en el robot para su posterior reproducción.

Figura 25. Pantalla LCD

El esquema de conexionado es el siguiente:

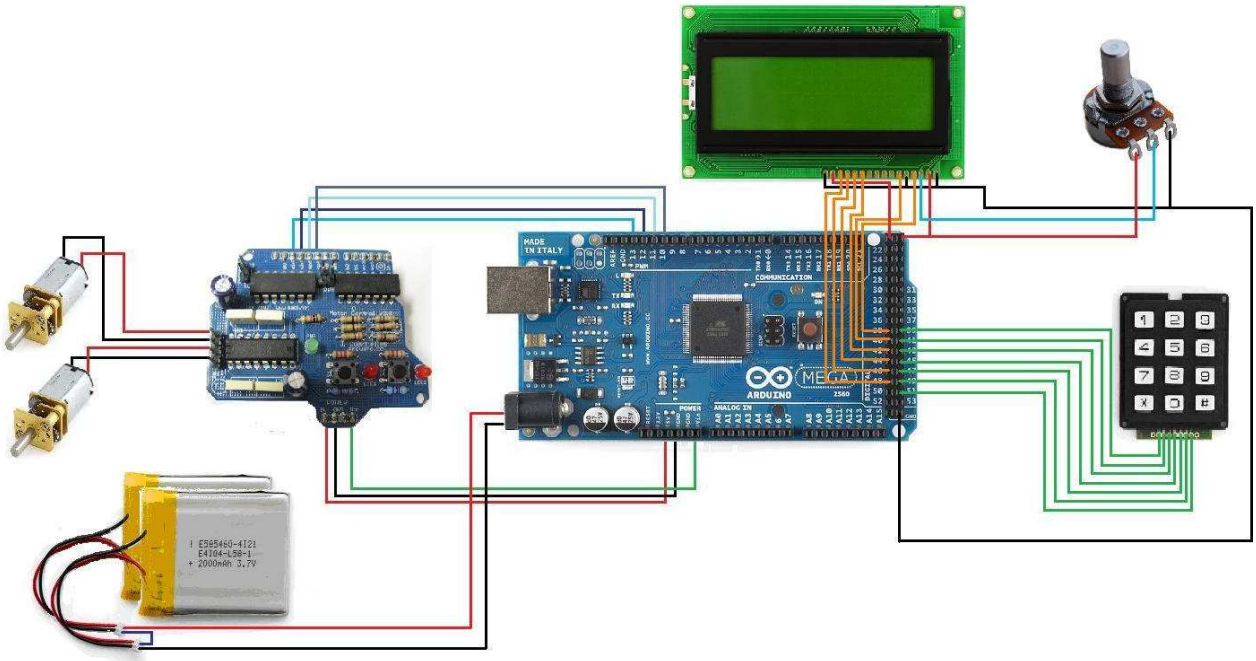


Figura 26. Esquema de conexionado del Bee-Bot UPNA.

Finalmente y tras una serie de ajustes del diseño tenemos un producto mucho más económico que el Bee-Bot comercial y que además de realizar las mismas acciones podemos manipular y obtener un gran número de variantes mejoradas. Teniendo en cuenta que es un dispositivo didáctico creemos que la posibilidad de entrar en la programación hace que se pueda utilizar para edades más avanzadas también.

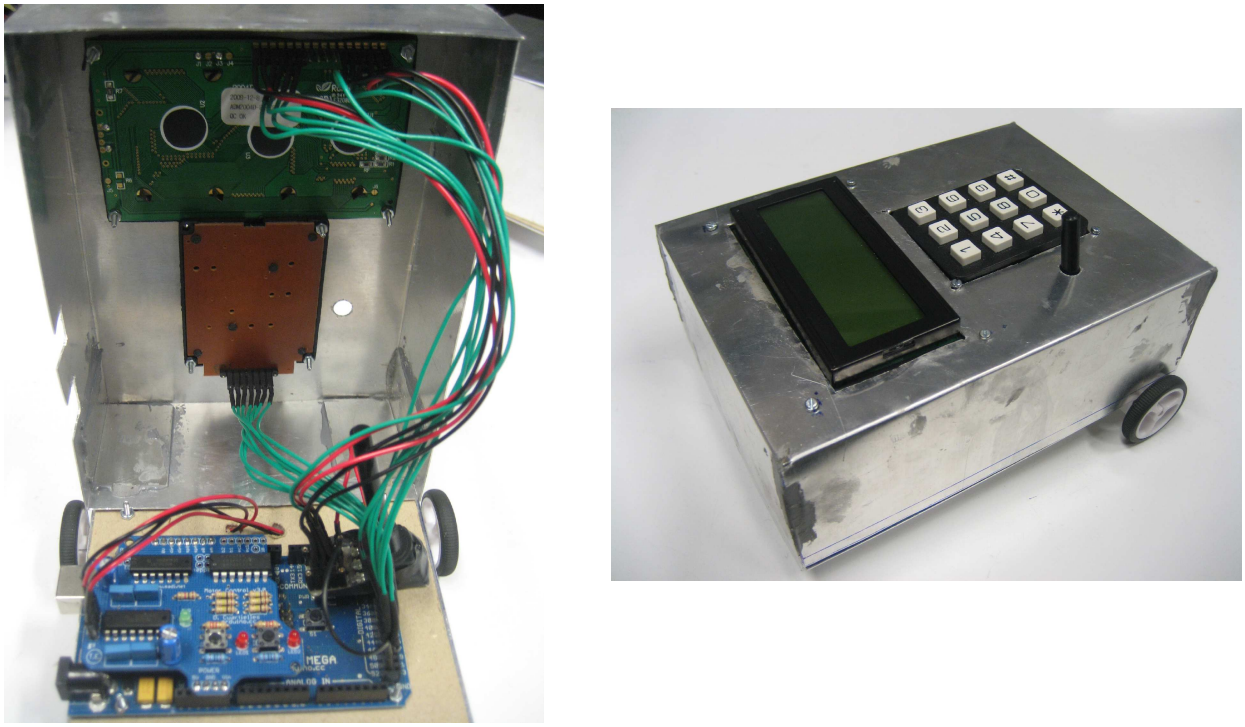


Figura 27. Bee-Bot, el esqueleto a la izquierda y el robot con carcasa en la parte derecha

Figura 28. Prototipo Bee-Bot finalizado

4.2 Validación de los diseños.

Debemos tener en cuenta, el espació por donde se va a utilizar nuestro robot. Una vez realizado el ensamblaje y la programación básica del funcionamiento de los robots, hemos procedido a la realización de una serie de pruebas de funcionamiento para que los robots estén perfectamente adecuados al entorno de trabajo.

4.3 Aplicaciones del proyecto.

La aplicación de nuestro proyecto es la de facilitar unos conocimientos básicos sobre el entorno Arduino, aunque a continuación se relatan algunas aplicaciones para las que nuestro robot podría ser una solución cuyos ejemplos se muestran en el punto de antecedentes del proyecto.

-Robots destinados a la seguridad: Tanto en interiores como en exteriores.
-Robots destinados a la exploración (en lugares de difícil acceso o donde el acceso comporte mucho riesgo para el hombre).
-Robots destinados a la vigilancia de bebés o personas incapacitadas.

En conclusión podemos decir que, basándonos en sus características técnicas, los robots no sólo se podrían limitar al uso para el que han sido diseñados sino que también como hemos podido comprobar serían aptos para múltiples aplicaciones puesto que se pueden unir, ampliar y programar en función del uso que deseemos atribuirles.

4.4 Ámbitos de utilización

Estos robots están principalmente diseñados, debido a sus características técnicas, para desarrollar sus funciones en un espacio interior, aunque no se descarta su utilización en un medio exterior siempre y cuando las condiciones meteorológicas sean las adecuadas y nos encontremos en un entorno sin demasiados obstáculos o desniveles pronunciados.

La relación de reducción de los motores que lleva incorporado nuestro robot (30:1) es una relación de reducción baja que ha sido escogida básicamente para minimizar el consumo de los motores y desplazarse de esta manera de forma muy correcta en superficies interiores.

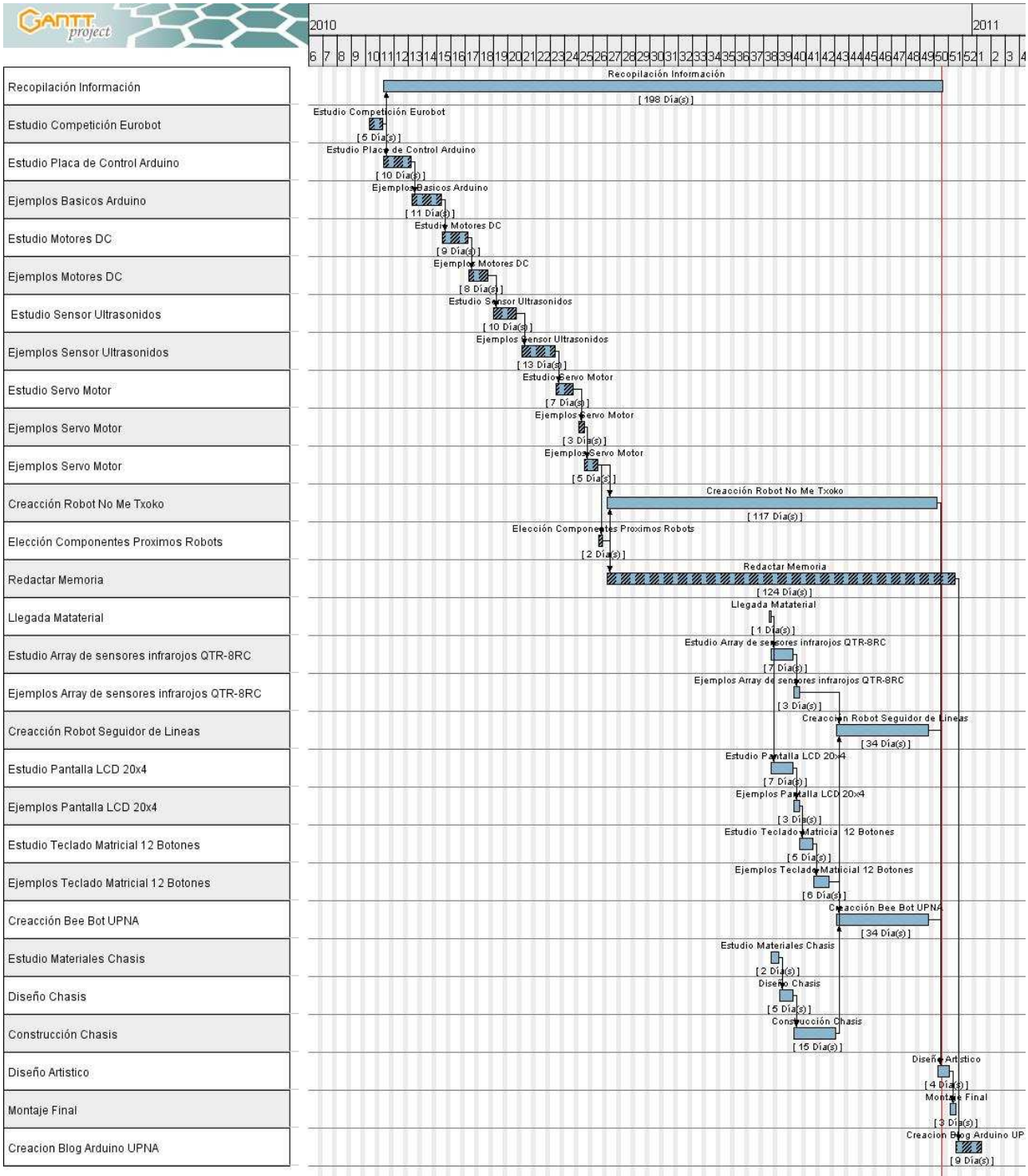
No obstante, si se pretendiera que la utilización de nuestro robot fuera para exteriores se podría realizar la adaptación al medio, de una forma muy sencilla, puesto que existen motores comercializados por Pololu, con exactamente las mismas dimensiones que los que nosotros hemos instalado (que nos permitirían ser instalados sin problemas en nuestro chasis sin tener que hacerle ninguna modificación) pero con una relación de reducción mucho más grande, concretamente de hasta 298:1 con lo que tendríamos una fuerza de torsión de hasta 6,5 Kg./cm. En contraposición de nuestros 0,3 Kg./cm. Pero eso si, sacrificaríamos en mucho la velocidad de nuestro robot y de igual forma aumentaríamos considerablemente el consumo de corriente, lo que disminuiría bastante la vida útil de nuestra batería.

En todo caso, como vemos, el problema del terreno se podría solventar fácilmente pero lo que no sería tan fácil de solventar sería el encapsulado de los componentes internos y externos de nuestro robot, para adaptarlo a condiciones metereológicas adversas, como la lluvia que podría dañar sin lugar a dudas, los componentes electrónicos de nuestro dispositivo.

5. Comentarios Finales.

5.1 Plan de trabajo.

En nuestro caso creamos un diagrama de Gantt para establecernos unos plazos con fechas límites de todas las tareas que debíamos realizar para crear nuestro proyecto.



5.2 Lista de materiales.

Lista de Materiales	
Electrónica	Unidades
Arduino Duemilanove USB	2
Arduino MEGA	1
Cargador de Baterías Lipo	1
Arduino Motor Shiel Kit	3
Sensores	Unidades
Sensor de distancia por ultrasonidos PING	1
Array de sensores infrarrojos QTR-8RC	1
Actuadores	Unidades
Servomotor estándar S03N	1
Motor Micro Metal DC con reductora 30:1	6
Componentes Mecánicos	Unidades
Rueda de goma 32x7mm (2 und)	3
Rueda loca metálica 3/8	3
Chasis Chapa de Hierro	1
Tornillería	Unidades
Tornillos Métrica 2	1
Componentes electrónicos	Unidades
Batería Lipo 1100mAh	1
Batería Lipo 2000mAh	3
Batería AA recargable - NiMH 2500mAh	4
Pantalla LCD 20x4 caracteres – Negro/Verde	1
Teclado Matricial de 12 botones	1
Cables conexión 0,2mm Negro	1
Cables conexión 0,2mm Verde	1
Cables conexión 0,2mm Rojo	1
Potenciómetro 10K	1
Varios	Unidades
Estaño Soldar	1
Tira de pines Hembra 16	2
Tira de pines Macho 16	2
Tubo Termorretráctil 2,4mm Negro	1
Pegatinas	5
Pegamento Nural 21	2
Bisagra	1
Base para baterías (4xAA)	1
Horas de Trabajo	200

5.3 Presupuesto.

Presupuesto			
Electrónica	Unidades	Precio / Unidad	Total
Arduino Duemilanove USB	2	20	40
Arduino MEGA	1	41	41
Cargador de Baterías Lipo	1	13,6	13,6
Arduino Motor Shiel Kit	3	21	63
Sensores	Unidades	Precio / Unidad	Total
Sensor de distancia por ultrasonidos PING	1	27,3	27,3
Array de sensores infrarrojos QTR-8RC	1	11,9	11,9
Actuadores	Unidades	Precio / Unidad	Total
Servomotor estándar S03N	1	11,9	11,9
Motor Micro Metal DC con reductora 30:1	6	13,2	79,2
Componentes Mecánicos	Unidades	Precio / Unidad	Total
Rueda de goma 32x7mm (2 und)	3	5,95	17,85
Rueda loca metálica 3/8	3	3,15	9,45
Chasis Chapa de Hierro	1	30	30
Tornillería	Unidades	Precio / Unidad	Total
Tornillos Métrica 2	1	0,95	0,95
Componentes electrónicos	Unidades	Precio / Unidad	Total
Batería Lipo 1100mAh	1	9,9	9,9
Batería Lipo 2000mAh	3	13,55	40,65
Batería AA recargable - NiMH 2500mAh	4	2,5	10
Pantalla LCD 20x4 caracteres – Negro/Verde	1	15,1	15,1
Teclado Matricial de 12 botones	1	5,2	5,2
Cables conexión 0,2mm Negro	1	3,19	3,19
Cables conexión 0,2mm Verde	1	3,19	3,19
Cables conexión 0,2mm Rojo	1	3,19	3,19
Potenciómetro 10K	1	1,25	1,25
Varios	Unidades	Precio / Unidad	Total
Estaño Soldar	1	7,4	7,4
Tira de pines Hembra 16	2	1,25	2,5
Tira de pines Macho 16	2	1,25	2,5
Tubo Termorretráctil 2,4mm Negro	1	3,6	3,6
Pegatinas	5	4,5	22,5
Pegamento Nural 21	2	12,6	25,2
Bisagra	1	1,5	1,5
Base para baterías (4xAA)	1	2	2
Horas de Trabajo	200	11,52	2304
		Precio Total	2809,02

5.4 Objetivos conseguidos.

Finalmente hemos conseguido los siguientes objetivos:

-Integrar los siguientes elementos a nuestro robot: Placas controladoras Arduino, sensor seguidor de línea, placas controladoras de motores, sensor de ultrasonidos, pulsadores, motores DC y servo motores, pantallas LCD y teclados numéricos.

-Diseño de un chasis adecuado para robots.

-Elaboración de programas básicos para probar el buen funcionamiento del diseño.

-Construcción física de varios prototipos.

-Conexión de la placa ATMEGA con comunicación serie.

- Repositorio de materiales para la iniciación al entorno Arduino y a concursos de robótica.

5.5 Conclusiones.

El controlador ATMEGA168, basado en la plataforma Arduino, ofrece prestaciones óptimas, tanto en su capacidad para la conexión de sensores y actuadores, como en la facilidad de su programación, utilizando entornos y lenguajes de programación de alto nivel; además de ser la placa controladora más utilizada en concursos de robótica en la actualidad.

A pesar de su alto nivel de programación encontramos distintos fallos producidos por la no correcta interpretación del código por parte de la placa controladora de Arduino, así como otras imperfecciones en la precisión de los movimientos de nuestros robots.

Es necesaria una valoración y estudio exhaustivo en cuanto a los elementos mecánicos se refiere, tales como coeficientes de rozamiento de los neumáticos, distancias entre ejes, tensiones…Más allá de la función estructural de contener los diversos elementos de los que consta el robot.

Una vez concluido el PFC han quedado distintos aspectos por perfeccionar. Estos aspectos son los siguientes:

- Mejorar la precisión.
- Mejorar la velocidad de los robots.
- Mejorar los programas para que se ejecuten en la mayor brevedad, con la mayor sencillez posible y realicen más movimientos.

5.6 Mejoras Futuras.

Una de las mejoras que se podrían incluir en nuestro robot es la creación de un modo teledirigido en lo que sería una posible ampliación de este proyecto, así como el diseño de un entorno único donde se podrían integrar todos los comandos de radio control y telemetría del robot para poder controlarlo a distancia si así lo requiere.

Por otro lado otra posible mejora futura bastante interesante sería la de incorporar al robot la propiedad de saber cuando se le esta acabando la batería y que éste de forma automática se fuera directo a su propia estación de carga y se recargara de forma autónoma. Esta característica ya la podemos encontrar en diversos modelos del mercado relatados en el punto de antecedentes y la verdad es que con esta característica incorporada el usuario del servicio se podría despreocupar totalmente del robot puesto que no debería prestarle ningún tipo de atención al mismo.

También se le podría incorporar algún tipo de dispositivo al robot, que tuviera dos funciones una la de permitirle al tener localizado a otros robots (mediante algún tipo de localizador de infrarrojos o mecanismo similar) y por lo tanto de esta manera poder ir donde ellos o evitarlos, sin tener que rastrear la estancia.

Otra vía de mejora e investigación sería el tema de brazos mecánicos que nos den la posibilidad de interactuar más activamente con el entorno.

Sin lugar a dudas si damos rienda suelta a nuestra imaginación las posibles mejoras y ampliaciones que podríamos incluir en nuestro proyecto son infinitas siempre y cuando los medios tecnológicos de los que disponemos en la actualidad nos lo permitieran.

6. Bibliografía.

-Apuntes de Servomotores del departamento de física, ingeniería de sistemas y teoría de la señal de la Universidad de Alicante. Autores: Francisco A. Candelas Herías y Juan A. Corrales Ramón.

-Documentación del curso de robótica "JETBOT" de la E.T.S.E.I.T. Autores: Arnau Bellavista y Ferràn Gascón.

[1] http://www.Arduino.cc (Entorno Arduino)
[2] http://www.freeduino.org/ (Entorno Arduino)
[3] http://www.ladyada.net/learn/lcd/charlcd.html (Componentes y robótica en general)
[4] http://www.hispavila.com/3ds/atmega/pulsadores.html (Componentes y robótica en general)
[5] http://www.Pololu.com (Componentes y robótica en general)
[6] http://www.adafruit.com/forums/viewtopic.php?f=8&t=14423 (Foro entorno Arduino)
[7] http://www.arduinobot.pbworks.com/w/page/10175779/Motores-DC (Componentes Electrónicos)
[8] http://www.wikipedia.org/ (Enciclopedia electrónica)

ANEXOS.

Artículos en referencia a la Justificación del proyecto.

A. Manual de instrucciones uso de Baterías Li-Po.

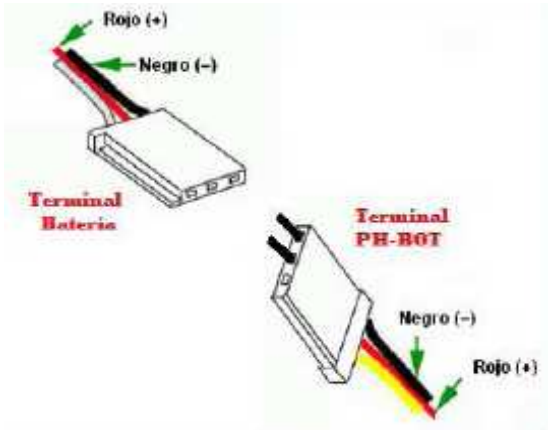
Los pasos a seguir para poner en funcionamiento el PH-BOT son muy sencillos.

1-Situar el PH-BOT en la estación de reposo.
2-Encender el interruptor general situado en la parte inferior trasera del robot.
3-Encender el pulsador verde situado en la parte superior delantera.
4-Comprobar el estado del robot mirando el mensaje de la pantalla LCD.

Con el mensaje PH-BOT Ready, sabríamos que el robot esta funcionando correctamente y que por tanto quedaría en espera, para realizar sus funciones.

En caso de que en la pantalla no apareciera nada, querría decir que el robot se ha quedado sin baterías y por tanto deberíamos proceder a su carga (mirar siguiente apartado) antes de volverlo a poner en funciona-miento.

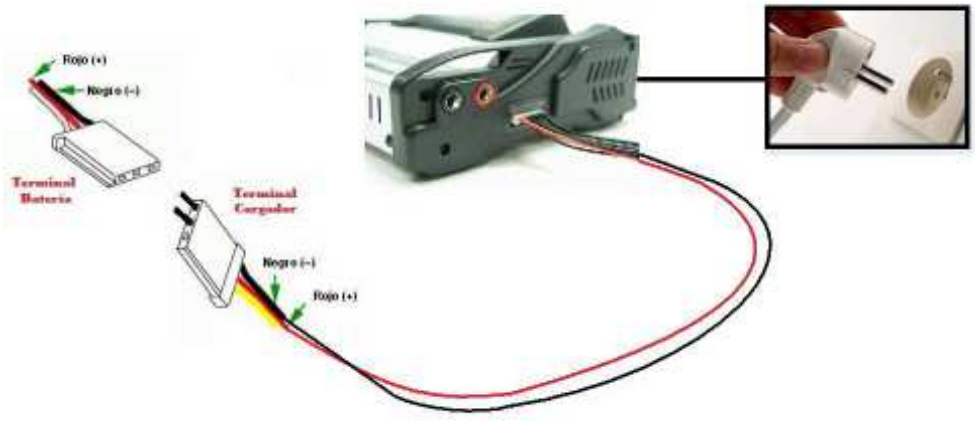
Instrucciones para la carga de las baterías.



Antes de proceder a la carga de las baterías deberemos desenchufar el terminal de la batería con respecto al terminal del PH-BOT tal y como vemos en la figura.

Una vez desenchufados los terminales cogeremos el terminal de la batería y lo enchufaremos al terminal del cargador. Al terminar la carga el cargador nos avisará mediante una luz fija de color rojo.

ADVERTENCIA: ES MUY IMPORTANTE RESPETAR LA POLARIDAD DURANTE LA CONEXIÓN DE LOS TERMINALES. EN CASO DE NO RESPETAR LA POLARIDAD LA BATERÍA PODRIA EXPLOTAR.



Normas de uso de las Baterías LIPO:

· Cerciórese de utilizar un cargador capaz de cargar baterías de Polímero de Litio. No utilice otro tipo de cargador.

· Cargar en un contenedor o zona ignifuga. Nunca en el modelo o en el interior de un vehiculo.

· Asegúrese perfectamente de programar correctamente el cargador para el pack que se va a cargar tanto en voltaje como en intensidad.

· No cargue sus baterías de Polímero de Litio por con intensidad mayor a 1C. Hacerlo reducirá la capacidad y vida de sus baterías con muy poco ahorro en tiempo de carga. Pueden cargarse por debajo de 1C es incluso recomendable. Recomendamos una intensidad de carga de entre 0,2C y 0,7C.

· Inspeccione cuidadosamente el pack especialmente sí el modelo ha sufrido un accidente. Si esta deformado no lo utilice y deshágase de él.

· No descargue los elementos por debajo de 3V durante la descarga, (6V para un pack de 2S, 9V para uno de 3S, etc.) una sola vez que se supere este limite causara daños irreparables en el elemento/pack. Utilice siempre un regulador con programación (corte) para Lipo o un avisador de voltaje mínimo en el modelo o ambas cosas; para usos de baja descarga se debe instalar un PCM.

Este voltaje mínimo puede ser superior o inferior dependiendo del tipo de pack, el fabricante del mismo debe proporcionarle ese dato.

· La vida de su pack será mucho más larga si en cada descarga/utilización no se llega a descargar completamente el pack.

· No utilizar los elementos/packs por encima de su descarga máxima (Indicada en el pack)

· No golpee, doble o deforme el pack de ningún modo.

· No seguir utilizando ningún elemento/pack que halla incrementado su volumen (parecido a un globo)

· Las baterías de Lipo no deben exceder 60ºC/140ºF durante su uso, si fuera así indicaría que el pack no es el idóneo.

· No montar packs de elementos/packs de capacidad desconocida o diferente en serie y/o paralelo.

· Mantenga sus baterías donde niños o animales no puedan acceder.

· Se debe tener siempre mucho cuidado de no cortocircuitar los elementos/packs de Lipo.

· Si el electrolito que tiene la batería en su interior toca su piel lavarla con abundante agua y jabón. Si entrase en sus ojos lávelos con agua fría y busque ayuda médica.

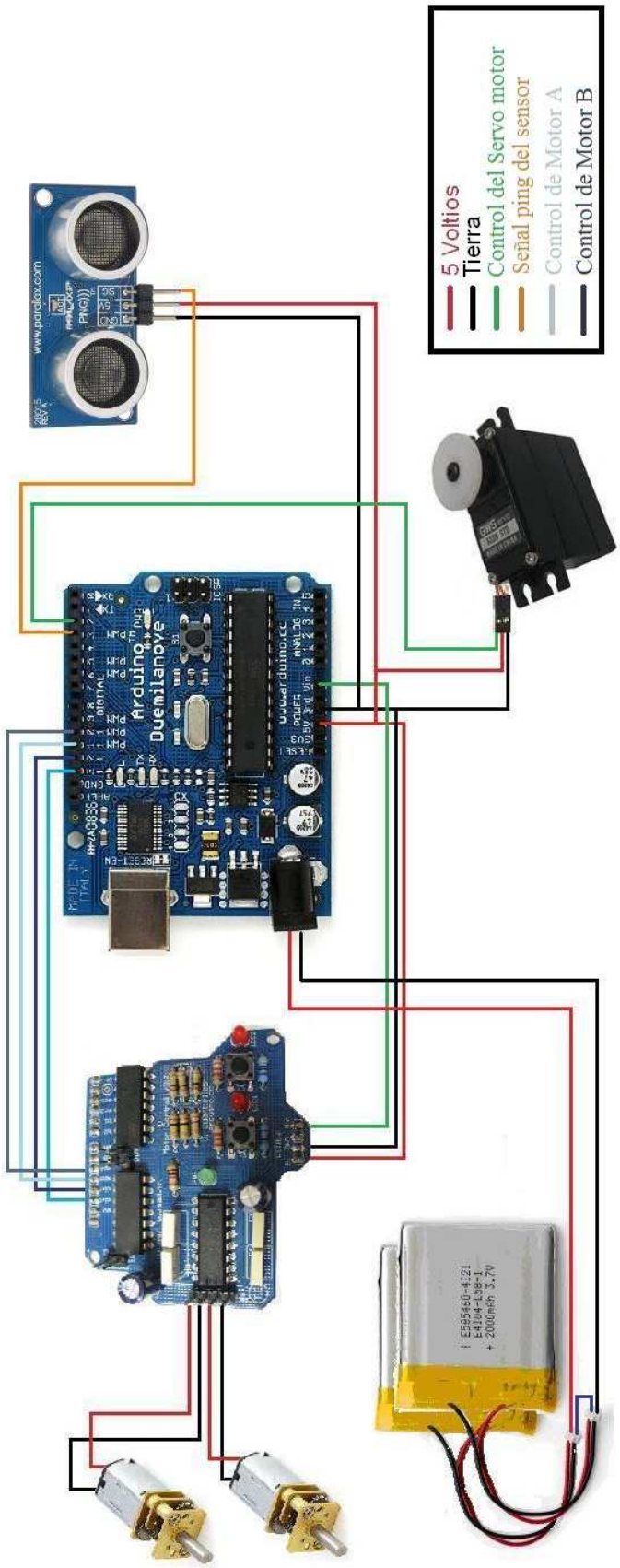
· Se debe tener siempre mucho cuidado de no cortocircuitar los elementos /packs de Lipo. Si eso ocurre la corriente que pasa a través de los terminales, cables o conectores los sobrecalentara en una fracción de segundo y es posible que las partes aislantes de los cables se funda si son de PVC, debido a eso siempre recomendamos el uso de cable con funda de silicona cuando se montan packs. Además de eso un cortocircuito severo dañara el pack irremediablemente.

· Cuando no vaya a utilizar las baterías de Polímero de Litio guárdelas a media carga (3,7/3,9V), nunca vacías o completamente cargadas.

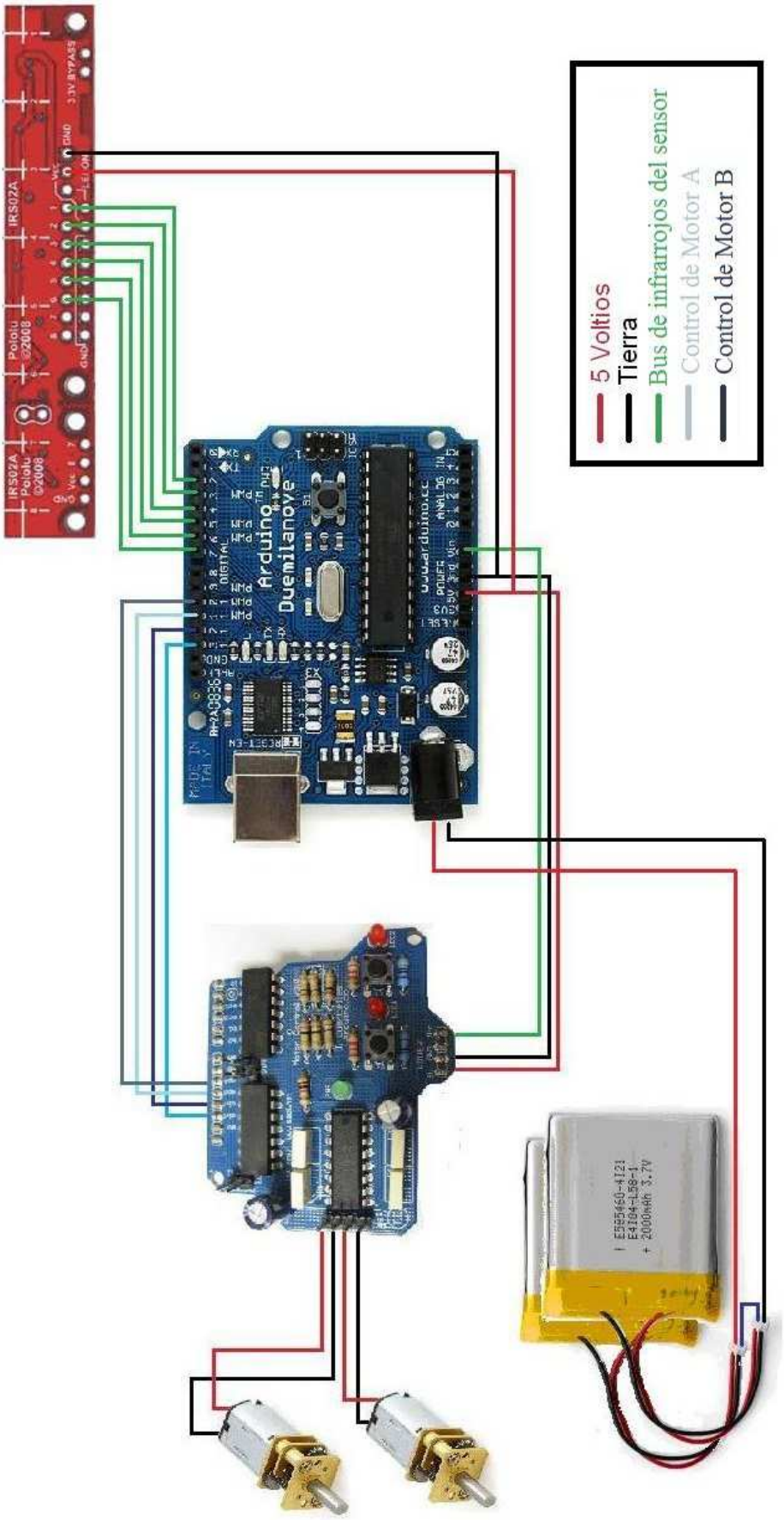
Siguiendo estas normas y recomendaciones básicas los packs tendrán una vida larga y nuestro modelo funcionará con potencia y altas prestaciones.

B. Esquema de conexiones.

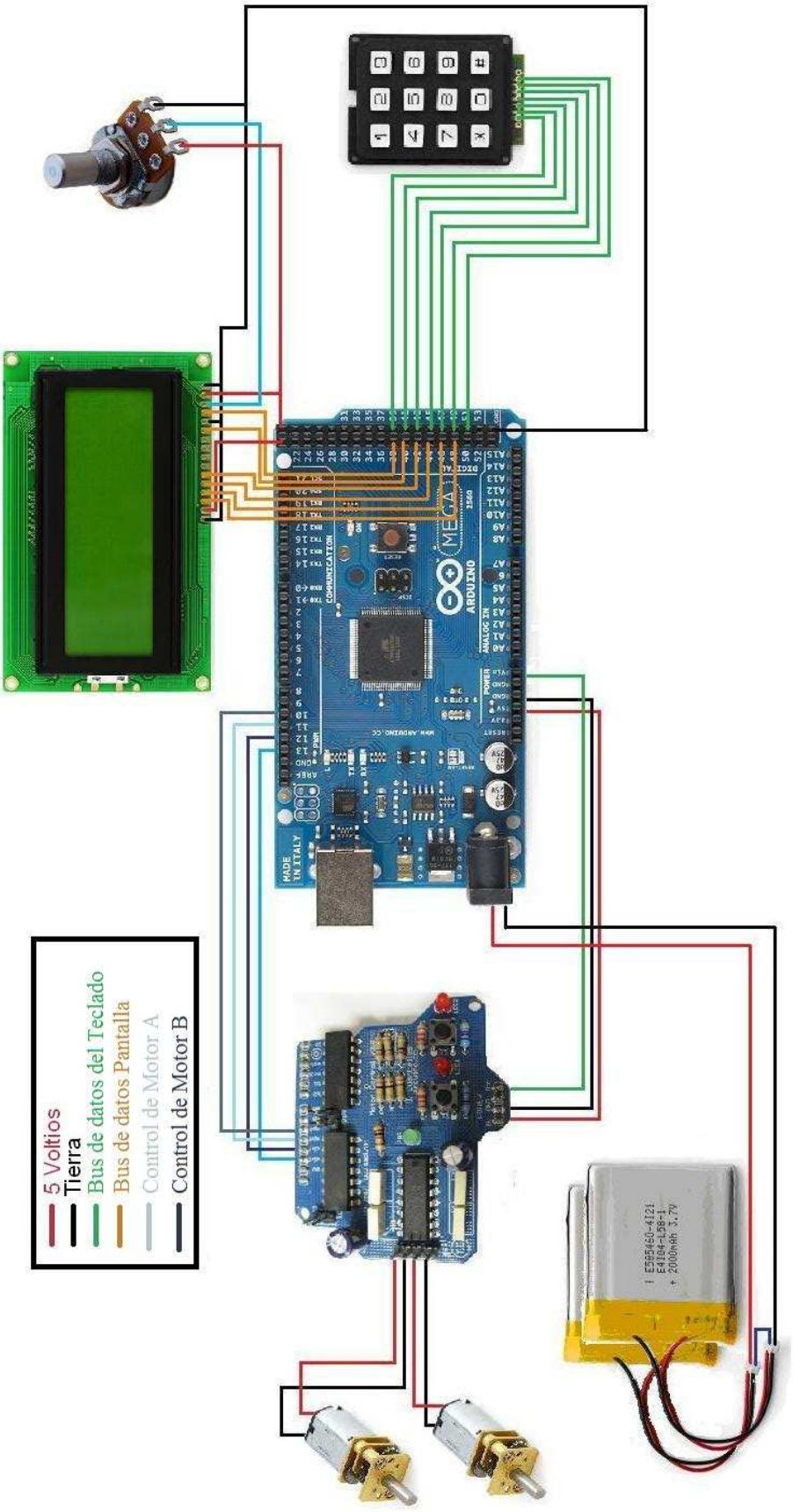
ROBOT 1: **"NoMeTxoko" UPNA**



ROBOT 2: **Seguidor de Líneas UPNA**



ROBOT 3: **Bee-Bot UPNA**



C. Listado de programas.

Programas Test
- Test de Movimiento de Motores DC
- Test de Velocidad de Motores DC
- Test de Servo Motores
- Test del Sensor de Ultrasonidos
- Test del Sensor de Infrarrojos
- Test de la Pantalla LCD
- Test del Teclado Numérico

Programas Robots
- "NoMeTxoko" UPNA
- Seguidor de línea UPNA
- Bee-Bot UPNA

Test de Movimiento de Motores DC:

/* TEST MOVIMIENTO MOTORES DC */

```
//En este programa de test pretendemos comprobar que los motores DC funcionan correctamente,
//por lo que vamos ha hacer que se muevan en ambas direcciones.
//Este ejemplo está hecho para dos motores DC, y una placa Arduino

//Primero asignamos los pines de dirección y velocidad para ambos motores
int dirA = 12;              //asignamos la dirección del motor A al pin 12 de la placa Arduino
int dirB = 13;    //asignamos la dirección del motor B al pin 13 de la placa Arduino
int speedA = 10;            //asignamos la velocidad del motor A al pin 10 de la placa Arduino
int speedB = 11;            //asignamos la dirección del motor B al pin 11 de la placa Arduino


void setup()
{
  Serial.begin(9600);          // establecemos la librería Serie a 9600 bps
  Serial.println("¡Test de Movimiento para Motores DC!");   // mostramos en pantalla
  pinMode (dirA, OUTPUT);       // ponemos como salida el pin asignado a dirA
  pinMode (dirB, OUTPUT);       // ponemos como salida el pin asignado a dirB
  pinMode (speedA, OUTPUT);  // ponemos como salida el pin asignado a speedA
  pinMode (speedB, OUTPUT);  // ponemos como salida el pin asignado a speedB
}


void loop()
{
  //1: ADELANTE + PARADA
  // movemos ambos motores en la misma dirección para que siga para adelante
  Serial.print(" arranca");          // mostramos en pantalla la frase entrecomillada
  digitalWrite (dirA, HIGH);        // sacamos nivel alto por la dirA para que gire adelante
  digitalWrite (dirB, HIGH);        // sacamos nivel alto por la dirB para que gire adelante
  digitalWrite (speedA, 135);       // ponemos 135 de velocidad de giro del motor A
  digitalWrite (speedB, 135);       // ponemos 135 de velocidad de giro del motor B

  delay(1000); // mantiene los motores girando durante 1000ms

  // para los motores A y B
  Serial.print(" parada"); // mostramos en pantalla la frase entrecomillada
  digitalWrite (speedA, 0);         // ponemos 0 de velocidad de giro del motor A
  digitalWrite (speedB, 0);         // ponemos 0 de velocidad de giro del motor B

  delay(100); // mantiene los motores parados durante 100ms
```

```
  //2: ATRÁS + PARADA
  // mueve hacia atrás los motores A y B
  Serial.print(" atras");    // mostramos en pantalla la frase entrecomillada
  digitalWrite (dirA, LOW);        // sacamos nivel bajo por la dirA para que gire hacia atrás
  digitalWrite (dirB, LOW);        // sacamos nivel bajo por la dirB para que gire hacia atrás
  digitalWrite (speedA, 135);      // ponemos 135 de velocidad de giro del motor A
  digitalWrite (speedB, 135);      // ponemos 135 de velocidad de giro del motor B

  delay(1000); // mantiene los motores girando durante 1000ms

  // para los motores A y B
  Serial.print(" parada"); // mostramos en pantalla la frase entrecomillada
  digitalWrite (speedA, 0);        // ponemos 0 de velocidad de giro del motor A
  digitalWrite (speedB, 0);        // ponemos 0 de velocidad de giro del motor B

  delay(100); // mantiene los motores parados durante 100ms

  //3: IZQUIERDA + PARADA
  // motor A hacia delante y el B hacia atrás - gira a la izquierda
  Serial.print(" izquierda");      // mostramos en pantalla la frase entrecomillada
  digitalWrite (dirA, HIGH);       // sacamos nivel alto por la dirA para que gire adelante
  digitalWrite (dirB, LOW);        // sacamos nivel bajo por la dirB para que gire hacia atrás
  digitalWrite (speedA, 135);      // ponemos 135 de velocidad de giro del motor A
  digitalWrite (speedB, 135);      // ponemos 135 de velocidad de giro del motor B

  delay(1000); // mantiene los motores girando durante 1000ms

  // para los motores A y B
  Serial.print(" parada"); // mostramos en pantalla la frase entrecomillada
  digitalWrite (speedA, 0);        // ponemos 0 de velocidad de giro del motor A
  digitalWrite (speedB, 0);        // ponemos 0 de velocidad de giro del motor B

  delay(100); // mantiene los motores parados durante 100ms

  //4: DERECHA + PARADA
  // motor A hacia atrás y el B hacia delante - gira a la derecha
  Serial.print(" derecha");        // mostramos en pantalla la frase entrecomillada
  digitalWrite (dirA, LOW);        // sacamos nivel bajo por la dirA para que gire hacia atrás
  digitalWrite (dirB, HIGH);       // sacamos nivel alto por la dirB para que gire adelante
  digitalWrite (speedA, 135);
  digitalWrite (speedB, 135);

  delay(1000); // mantiene los motores girando durante 1000ms

  // para los motores A y B
  Serial.print(" parada"); // mostramos en pantalla la frase entrecomillada
  digitalWrite (speedA, 0);        // ponemos 0 de velocidad de giro del motor A
  digitalWrite (speedB, 0);        // ponemos 0 de velocidad de giro del motor B

  delay(100); // mantiene los motores parados durante 100ms
}
```

Test de Velocidad de Motores DC:

//*TEST VELOCIDAD MOTORES DC*//

```
//En este programa de test pretendemos comprobar que los motores DC funcionan correctamente,
//por lo que vamos ha hacer que se muevan en una dirección pero a distintas velocidades.
//Utilizamos la función analogwrite (variable de velocidad, valor de velocidad) para controlar
//la velocidad de los motores, ya que le damos valores analógicos de 0 a 255. Podemos
//cambiar la dirección de los motores cambiando HIGH (adelante) por LOW (atrás).Este ejemplo
//está hecho para dos motores DC, y una placa Arduino

//Primero asignamos los pines de dirección y velocidad para ambos motores
int dirA = 12;              //asignamos la dirección del motor A al pin 12 de la placa Arduino
int dirB = 13;    //asignamos la dirección del motor B al pin 13 de la placa Arduino
int speedA = 10;            //asignamos la velocidad del motor A al pin 10 de la placa Arduino
int speedB = 11;            //asignamos la dirección del motor B al pin 11 de la placa Arduino

void setup()
{
  Serial.begin(9600);          // establecemos la librería Serie a 9600 bps
  Serial.println("¡Test de Velocidad para Motores DC!");   // mostramos en pantalla
  pinMode (dirA, OUTPUT);       // ponemos como salida el pin asignado a dirA
  pinMode (dirB, OUTPUT);       // ponemos como salida el pin asignado a dirB
  pinMode (speedA, OUTPUT);  // ponemos como salida el pin asignado a speedA
  pinMode (speedB, OUTPUT);  // ponemos como salida el pin asignado a speedB
}

void loop()
{
  //1: VELOCIDAD "0" -- MÍNIMO
  // movemos ambos motores en la misma dirección para que siga para adelante
  Serial.print(" Velocidad 0");      // mostramos en pantalla la frase entrecomillada
  digitalWrite (dirA, HIGH);        // sacamos nivel alto por la dirA para que gire adelante
  digitalWrite (dirB, HIGH);        // sacamos nivel alto por la dirB para que gire adelante
  analogWrite (speedA, 0);          // ponemos 0 de velocidad de giro del motor A
  analogWrite (speedB, 0);          // ponemos 0 de velocidad de giro del motor B

  delay(1000); // mantiene los motores girando durante 1000ms

  // para los motores A y B
  Serial.print(" parada"); // mostramos en pantalla la frase entrecomillada
  digitalWrite (speedA, 0);         // ponemos 0 de velocidad de giro del motor A
  digitalWrite (speedB, 0);         // ponemos 0 de velocidad de giro del motor B

  delay(100); // mantiene los motores girando durante 100ms

  //2: VELOCIDAD "100"
  // movemos ambos motores en la misma dirección para que siga para adelante
  Serial.print (" Velocidad 100"); // mostramos en pantalla la frase entrecomillada
  digitalWrite (dirA, HIGH);        // sacamos nivel alto por la dirA para que gire adelante
  digitalWrite (dirB, HIGH);        // sacamos nivel alto por la dirB para que gire adelante
  analogWrite (speedA, 100);     // ponemos 100 de velocidad de giro del motor A
  analogWrite (speedB, 100);     // ponemos 100 de velocidad de giro del motor B

  delay(1000); // mantiene los motores girando durante 1000ms

  // para los motores A y B
  Serial.print (" parada");          // mostramos en pantalla la frase entrecomillada
  digitalWrite (speedA, 0);          // ponemos 0 de velocidad de giro del motor A
  digitalWrite (speedB, 0);          // ponemos 0 de velocidad de giro del motor B
```

```
delay(100); // mantiene los motores girando durante 100ms
//3: VELOCIDAD "135"
// movemos ambos motores en la misma dirección para que siga para adelante
Serial.print (" Velocidad 135"); // mostramos en pantalla la frase entrecomillada
digitalWrite (dirA, HIGH);       // sacamos nivel alto por la dirA para que gire adelante
digitalWrite (dirB, HIGH);       // sacamos nivel alto por la dirB para que gire adelante
analogWrite (speedA, 135);       // ponemos 135 de velocidad de giro del motor A
analogWrite (speedB, 135);       // ponemos 135 de velocidad de giro del motor B

delay(1000); // mantiene los motores girando durante 1000ms

// para los motores A y B
Serial.print (" parada");        // mostramos en pantalla la frase entrecomillada
digitalWrite (speedA, 0);        // ponemos 0 de velocidad de giro del motor A
digitalWrite (speedB, 0);        // ponemos 0 de velocidad de giro del motor B

delay(100); // mantiene los motores girando durante 100ms

//4: VELOCIDAD "150"
// movemos ambos motores en la misma dirección para que siga para adelante
Serial.print (" Velocidad 150"); // mostramos en pantalla la frase entrecomillada
digitalWrite (dirA, HIGH);       // sacamos nivel alto por la dirA para que gire adelante
digitalWrite (dirB, HIGH);       // sacamos nivel alto por la dirB para que gire adelante
analogWrite (speedA, 150);       // ponemos 150 de velocidad de giro del motor A
analogWrite (speedB, 150);       // ponemos 150 de velocidad de giro del motor B

delay(1000); // mantiene los motores girando durante 1000ms

// para los motores A y B
Serial.print(" parada"); // mostramos en pantalla la frase entrecomillada
digitalWrite (speedA, 0);        // ponemos 0 de velocidad de giro del motor A
digitalWrite (speedB, 0);        // ponemos 0 de velocidad de giro del motor B

delay(100); // mantiene los motores girando durante 100ms

//5: VELOCIDAD "175"
// movemos ambos motores en la misma dirección para que siga para adelante
Serial.print (" Velocidad 175"); // mostramos en pantalla la frase entrecomillada
digitalWrite (dirA, HIGH);       // sacamos nivel alto por la dirA para que gire adelante
digitalWrite (dirB, HIGH);       // sacamos nivel alto por la dirB para que gire adelante
analogWrite (speedA, 175);       // ponemos 175 de velocidad de giro del motor A
analogWrite (speedB, 175);       // ponemos 175 de velocidad de giro del motor B

delay(1000); // mantiene los motores girando durante 1000ms

// para los motores A y B
Serial.print (" parada");        // mostramos en pantalla la frase entrecomillada
digitalWrite (speedA, 0);        // ponemos 0 de velocidad de giro del motor A
digitalWrite (speedB, 0);        // ponemos 0 de velocidad de giro del motor B

delay(100); // mantiene los motores girando durante 100ms

//6: VELOCIDAD "200"
// movemos ambos motores en la misma dirección para que siga para adelante
Serial.print (" Velocidad 200"); // mostramos en pantalla la frase entrecomillada
digitalWrite (dirA, HIGH);       // sacamos nivel alto por la dirA para que gire adelante
digitalWrite (dirB, HIGH);       // sacamos nivel alto por la dirB para que gire adelante
analogWrite (speedA, 200);       // ponemos 200 de velocidad de giro del motor A
analogWrite (speedB, 200);       // ponemos 200 de velocidad de giro del motor B
```

```
  delay(1000); // mantiene los motores girando durante 1000ms

 // para los motores A y B
  Serial.print (" parada");          // mostramos en pantalla la frase entrecomillada
  digitalWrite (speedA, 0);          // ponemos 0 de velocidad de giro del motor A
  digitalWrite (speedB, 0);          // ponemos 0 de velocidad de giro del motor B

  delay(100); // mantiene los motores girando durante 100ms

  //7: VELOCIDAD "225"
  // movemos ambos motores en la misma dirección para que siga para adelante
  Serial.print (" Velocidad 225"); // mostramos en pantalla la frase entrecomillada
  digitalWrite (dirA, HIGH);         // sacamos nivel alto por la dirA para que gire adelante
  digitalWrite (dirB, HIGH);         // sacamos nivel alto por la dirB para que gire adelante
  analogWrite (speedA, 225);     // ponemos 225 de velocidad de giro del motor A
  analogWrite (speedB, 225);     // ponemos 225 de velocidad de giro del motor B

  delay(1000); // mantiene los motores girando durante 1000ms

  // para los motores A y B
  Serial.print (" parada");          // mostramos en pantalla la frase entrecomillada
  digitalWrite (speedA, 0);          // ponemos 0 de velocidad de giro del motor A
  digitalWrite (speedB, 0);          // ponemos 0 de velocidad de giro del motor B

  delay(100); // mantiene los motores girando durante 100ms

  //8: VELOCIDAD "250"
  // movemos ambos motores en la misma dirección para que siga para adelante
  Serial.print (" Velocidad 250"); // mostramos en pantalla la frase entrecomillada
  digitalWrite (dirA, HIGH);         // sacamos nivel alto por la dirA para que gire adelante
  digitalWrite (dirB, HIGH);         // sacamos nivel alto por la dirB para que gire adelante
  analogWrite (speedA, 250);     // ponemos 250 de velocidad de giro del motor A
  analogWrite (speedB, 250);     // ponemos 250 de velocidad de giro del motor B

  delay(1000); // mantiene los motores girando durante 1000ms

  // para los motores A y B
  Serial.print(" parada"); // mostramos en pantalla la frase entrecomillada
  digitalWrite (speedA, 0);          // ponemos 0 de velocidad de giro del motor A
  digitalWrite (speedB, 0);          // ponemos 0 de velocidad de giro del motor B

  delay(100); // mantiene los motores girando durante 100ms

  //9: VELOCIDAD "255" -- MÁXIMO
  // movemos ambos motores en la misma dirección para que siga para adelante
  Serial.print (" Velocidad 255"); // mostramos en pantalla la frase entrecomillada
  digitalWrite (dirA, HIGH);         // sacamos nivel alto por la dirA para que gire adelante
  digitalWrite (dirB, HIGH);         // sacamos nivel alto por la dirB para que gire adelante
  analogWrite (speedA, 255);     // ponemos 255 de velocidad de giro del motor A
  analogWrite (speedB, 255);     // ponemos 255 de velocidad de giro del motor B

  delay(1000); // mantiene los motores girando durante 1000ms

  // para los motores A y B
  Serial.print (" parada");          // mostramos en pantalla la frase entrecomillada
  digitalWrite (speedA, 0);          // ponemos 0 de velocidad de giro del motor A
  digitalWrite (speedB, 0);          // ponemos 0 de velocidad de giro del motor B

  delay(100); // mantiene los motores girando durante 100ms
}
```

Test de Servo Motores:

//*TEST SERVO MOTORES*//

```
//En este programa test pretendemos comprobar que los servo motores funcionan correctamente,
//por lo que vamos ha hacer realice un recorrido completo de izquierda a derecha con varias
//paradas. Este ejemplo está programado para un servo motor y una placa Arduino. Incluimos una
//librería para el control de los servo motores llamada "Servo.h". Realiza un movimiento de
//180º siendo 0º el mínimo y 180º el máximo, es mejor no llegar a los extremos por lo que se
//pone 1º y 179º como mínimo y máximo respectivamente


#include <Servo.h>

Servo myservo;  // creamos un objeto servo para controlar al servo motor
                // con esta librería se pueden controlar un máximo de 8 servo motores

void setup()
{
  Serial.begin(9600);               // establecemos la librería Serie a 9600 bps
  Serial.println("¡Test para Servo Motores!");   // mostramos en pantalla
  myservo.attach(6);                // vinculamos el servo del pin 2 con el objeto servo
}

void loop()
{

  // 1: LADO IZQUIERDO -- MÍNIMO
  // Movemos el servo motor
  Serial.print(" Grado: 1º");
  myservo.write(1);        // movemos el motor servo a los grados indicados
  delay(1500);          // esperamos 1500ms para que el servo alcance la posición

  // 2: DIAGONAL IZQUIERDA
  // Movemos el servo motor
  Serial.print(" Grado: 45º");
  myservo.write(45);       // movemos el motor servo a los grados indicados
  delay(1500);          // esperamos 1500ms para que el servo alcance la posición

  // 3: CENTRO
  // Movemos el servo motor
  Serial.print(" Grado: 90º");
  myservo.write(90);       // movemos el motor servo a los grados indicados
  delay(1500);          // esperamos 1500ms para que el servo alcance la posición

  // 4: DIAGONAL DERECHA
  // Movemos el servo motor
  Serial.print(" Grado: 135º");
  myservo.write(135);     // movemos el motor servo a los grados indicados
  delay(1500);          // esperamos 1500ms para que el servo alcance la posición

  // 5: LADO DERECH0 -- MÁXIMO
  // Movemos el servo motor
  Serial.print(" Grado: 179º");
  myservo.write(179);     // movemos el motor servo a los grados indicados
  delay(1500);          // esperamos 1500ms para que el servo alcance la posición

}
```

Test del Sensor de Ultrasonidos:

```
//*TEST SENSOR ULTRASONIDOS*//

//Para este test vamos a comprobar que realice bien la medida de distancia a un objeto el
//sensor de ultrasonidos. Este ejemplo está programado para un sensor de ultrasonidos
//PING)))™ Ultrasonic Distance Sensor (#28015). Envía un pulso cuadrado de 5us y espera a
//recibir la onda de vuelta, con lo que podemos calcular la distancia a la que se
//encuentra el objeto.


//Definimos el valor del pin de control sobre el ping del sensor de ultrasonidos
const int pingPin = 7;


void setup()
{
  Serial.begin(9600);          // establecemos la librería Serie a 9600 bps
  Serial.println("¡Test del Sensor de Ultrasonidos!");   // mostramos en pantalla
}



void loop()
{

  long duration, cm;                  // definimos la variable que va a estar en centímetros

  // Vamos a lanzar un pulso por el sensor de ultrasonidos para después recogerlo y así poder
  // calcular la distancia que existe entre en sensor y lo que tiene delante (max. 3m)
  pinMode(pingPin, OUTPUT);   // ponemos el pin de control del pulso del sensor como salida
  digitalWrite(pingPin, LOW);     // mandamos el comienzo de nivel bajo del pulso de 2ms
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);    // mandamos la parte central de nivel alto del pulso de 5ms
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);     // mandamos el final de nivel bajo del pulso

  pinMode(pingPin, INPUT);        // ponemos el pin del sensor como entrada

  duration = pulseIn(pingPin, HIGH);      // calculamos el tiempo de vuelta del pulso

  cm = microsecondsToCentimeters(duration);    // convertimos a distancia ese tiempo

  // Mostramos en pantalla el valor obtenido
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  delay(100);
}

//Función que convierte un valor en microsegundos a un valor en centímetros
long microsecondsToCentimeters(long microseconds)
{
   return microseconds / 29 / 2;
}
```

Test del Sensor de Infrarrojos:

/* TEST SENSOR INFRARROJOS */

```
//Hemos diseñado este programa para el test de control de sensores de infrarrojos con
//la intención de comprobar si el funcionamiento de los mismos es el esperado. Para
//ello hemos implementado este test que consiste en analizar si los valores obtenidos
//por los sensores es el adecuado.
//En este ejemplo se ha utilizado un array de sensores infrarrojos del modelo QTR-1RC
//y QTR-8RC, dos LEDs y una placa Arduino. Además incluye la librería de Pololu que
//aparece a continuación, y que está diseñada para seguidores de línea y sensores
//QTR-1RC y QTR-8RC.


#include <PololuQTRSensors.h>


/Definimos el valor de varias constantes que utiliza el programa test
#define NUM_SENSORS   8     // número de sensores utilizados
#define TIMEOUT       2500  // espera 2500us para apagar los sensores
#define QTR_NO_EMITTER_PIN  // el emisor no está controlado

//Los sensores del 0 al 7 están conectados a los pines digitales 3 a 10, respectivamente
PololuQTRSensorsRC qtrrc((unsigned char[]) {3, 4, 5, 6, 7, 8, 9, 10},  NUM_SENSORS, TIMEOUT);
unsigned int sensorValues[NUM_SENSORS];  //inicializamos una variable vacía


void setup()
{
  Serial.begin(9600);          // establecemos la librería Serie a 9600 bps
  Serial.println("¡Test para Sensores de Infrarrojos!");   // mostramos en pantalla
}


void loop()
{
  // 1: CALIBRACIÓN
  // Realizamos una calibración, es decir que recogemos la reflectividad que llega a cada
  // sensor y la guardamos. Después procesamos esa información y damos un valor medio de
  // mínima y máxima reflectividad obtenida por cada sensor
  int i;                  // inicializamos el contador i (contador de tiempo)
  pinMode(13, OUTPUT);              // ponemos el pin del LED como salida en el pin 13
  digitalWrite(13, HIGH);   // encendemos el LED para iniciar el modo de calibración

  for (i = 0; i < 400; i++) // hace que la calibración dure unos 10 segundos
  {
    qtrrc.calibrate();      // todos los sensores realizan 10 lecturas de 2500us cada una
  }
  digitalWrite(13, LOW);    // apagamos el LED para acabar el modo de calibración


  // 2: REPRESENTACIÓN
  // Ahora vamos a sacar por pantalla los valores obteneos durante la fase de calibración

  // Escribe en pantalla la media de los mínimos valores medidos para cada sensor
  for (i = 0; i < NUM_SENSORS; i++)
  {
    Serial.print(qtrrc.calibratedMinimumOn[i]);
    Serial.print(' ');
```

```
  }
  Serial.println();


 // Escribe en pantalla la media de los máximos valores medidos para cada sensor
  for (i = 0; i < NUM_SENSORS; i++)
  {
    Serial.print(qtrrc.calibratedMaximumOn[i]);
    Serial.print(' ');
  }

  // Añadimos dos líneas de separación
  Serial.println();
  Serial.println();
  delay(1000);

  // Indicamos con un encendido y apagado rápido que el programa ha finalizado
  digitalWrite(13, HIGH);   // encendemos el LED para iniciar el modo de calibración
  delay(1000);
  digitalWrite(13, LOW);    // apagamos el LED para acabar el modo de calibración
  delay(1000);
}
```

Test de la Pantalla LCD:

/* TEST PANTALLA LCD */

```
//En este programa de test se ha implementado para comprobar que todas las casillas de la
//pantalla funcionen correctamente. Este ejemplo está hecho para una pantalla LCD de 20x4,
//y una placa Arduino ATMega


#include <LiquidCrystal.h>


//La conexión de la pantalla con los pines de la placa es la siguiente
LiquidCrystal lcd(30,32, 34, 36, 38, 40);

int i=0;  // inicializamos el contador


void setup()
{
  Serial.begin(9600);              // Inicializa la comunicación serie
  lcd.begin(20, 4);                // Definimos el número de filas y columnas de la pantalla
}


//Vamos a ir recorriendo cada una de las 20x4 casillas representando en cada una de ellas un
//número del 0 al 9 consecutivamente.
void loop()
{
  for (i=1; i=4; i++)
  {
    for (j=1; i=20; j++)
    {
      for k=0; K=9; k++)
      {
            lcd.setCursor(i, j);       // Apuntamos a la casilla (i,j) de la pantalla LCD
        lcd.print(k);                  // Escribimos en la pantalla LCD la casilla a la que apunta
      }
    }
  }
}
```

Test del Teclado Numérico:

```
/* TEST TECLADO NUMÉRICO */

//El test pretende corroborar el teclado numérico envía correctamente la información al
//pulsar las teclas que contiene. Es un teclado matricial de 12 teclas distribuidas tal y
//como indica en el KEYMAP descrito en el programa.
//Este ejemplo se ha realizado para un teclado matricial de 12 teclas, y una placa Arduino.
//ATMega que contiene un número mayor de pines de conexión. Además incluye la librería que
//aparece a continuación, y que está diseñada para teclados numéricos matriciales como el
//que nosotros utilizamos.

#include <Keypad.h>

//Definimos un vector de 25 caracteres que va a contener los valores de las teclas que se
//hayan pulsado
char vector[25]={0};

int i=0;  // inicializamos el contador

const byte ROWS = 4;            //definimos que nuestro teclado va a tener cuatro filas
const byte COLS = 3;            //definimos que nuestro teclado va a tener tres columnas

//Definimos el KEYMAP, es decir cual es la posición matricial de nuestro teclado
char keys[ROWS][COLS] = {
  {1,2,3},
  {4,5,6},
  {7,8,9},
  {*,0,#}
};

//Conexión del keypad ROW0, ROW1, ROW2 y ROW3 a estos pins de Arduino.
byte rowPins[ROWS] = { 41, 51, 49, 45 };
//Conexión del keypad COL0, COL1 and COL2 a estos pins de Arduino.
byte colPins[COLS] = { 43, 39, 47 };
//Creamos el Keypad llamando a la siguiente función y con los valores ante asignados
Keypad kpd = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

void setup()
{
  Serial.begin(9600);           // Inicializa la comunicación serie
}

void loop()
{
  // Lectura de datos del keyboard (teclado)
  int key = kpd.getKey();

  // Ahora vamos hacer que se guarden y se escriban en pantalla
  if(key)  // es como if(key != NO_KEY)
  {
    switch (key)
    {
      case 1:
        vector[i]=key;        // Introducimos el valor de la pulsación en la posición i del vector
        Serial.print(vector[i]);   // Mostramos el valor de esa pulsación en pantalla
        i++;     // Sumamos 1 al puntero para que apunte a la siguiente posición del vector
      break;
      case 2:
        vector[i]=key;        // Introducimos el valor de la pulsación en la posición i del vector
```

```
      Serial.print(vector[i]);   // Mostramos el valor de esa pulsación en pantalla
      i++;     // Sumamos 1 al puntero para que apunte a la siguiente posición del vector
    break;

    case 3:
      vector[i]=key;        // Introducimos el valor de la pulsación en la posición i del vector
      Serial.print(vector[i]);   // Mostramos el valor de esa pulsación en pantalla
      i++;     // Sumamos 1 al puntero para que apunte a la siguiente posición del vector
    break;

    case 4:
      vector[i]=key;        // Introducimos el valor de la pulsación en la posición i del vector
      Serial.print(vector[i]);   // Mostramos el valor de esa pulsación en pantalla
      i++;     // Sumamos 1 al puntero para que apunte a la siguiente posición del vector
    break;

    case 5:
      vector[i]=key;        // Introducimos el valor de la pulsación en la posición i del vector
      Serial.print(vector[i]);   // Mostramos el valor de esa pulsación en pantalla
      i++;     // Sumamos 1 al puntero para que apunte a la siguiente posición del vector
    break;

    case 6:
      vector[i]=key;        // Introducimos el valor de la pulsación en la posición i del vector
      Serial.print(vector[i]);   // Mostramos el valor de esa pulsación en pantalla
      i++;     // Sumamos 1 al puntero para que apunte a la siguiente posición del vector
    break;

    case 7:
      vector[i]=key;        // Introducimos el valor de la pulsación en la posición i del vector
      Serial.print(vector[i]);   // Mostramos el valor de esa pulsación en pantalla
      i++;     // Sumamos 1 al puntero para que apunte a la siguiente posición del vector
    break;

    case 8:
      vector[i]=key;        // Introducimos el valor de la pulsación en la posición i del vector
      Serial.print(vector[i]);   // Mostramos el valor de esa pulsación en pantalla
      i++;     // Sumamos 1 al puntero para que apunte a la siguiente posición del vector
    break;

    case 9:
      vector[i]=key;        // Introducimos el valor de la pulsación en la posición i del vector
      Serial.print(vector[i]);   // Mostramos el valor de esa pulsación en pantalla
      i++;     // Sumamos 1 al puntero para que apunte a la siguiente posición del vector
    break;

    case *:
      vector[i]=key;        // Introducimos el valor de la pulsación en la posición i del vector
      Serial.print(vector[i]);   // Mostramos el valor de esa pulsación en pantalla
      i++;     // Sumamos 1 al puntero para que apunte a la siguiente posición del vector
    break;

    case #:
      vector[i]=key;        // Introducimos el valor de la pulsación en la posición i del vector
      Serial.print(vector[i]);   // Mostramos el valor de esa pulsación en pantalla
      i++;     // Sumamos 1 al puntero para que apunte a la siguiente posición del vector
    break;
   }
  }
}
```

PROGRAMAS ROBOTS:

Robot 1: No Me Txoko UPNA

/* NO ME TXOKO UPNA */

```
//Este ejemplo es un programa para arduino que utiliza la librería de arduino
//que aparece A continuación, y que está diseñada para la utilización de un
// servo motor. Además utiliza un sensor de ultrasonidos PING)))™.

#include <Servo.h>

//En esta ocasión utilizamos la placa Arduino AT168 Duemilanove la cual es
//suficiente para lo que necesitamos; es decir para el uso de dos motores DC,
//un servo motor y un sensor de ultrasonidos.

//Nuestra intención es que el robot se desplace por la estancia sin ningún
//recorrido marcado sin chocarse con ningún elemento de la misma:

//El primer bloque del programa va a consistir en un bucle a modo de
//Temporizador en el que está contenido todo el resto del programa, y que cada
//cierto tiempo se pare y compruebe que dirección seguir.
//Para la segunda fase del programa vamos a relizar el mismo sistema pero que
//se activará cuando la distancia a un objeto sea menor a un número de
//centímetros. Cuando se encuentre ese obstáculo realizará una comprobación a
//Cada lado el sensor de ultrasonidos, y decidirá continuar en la dirección en
//la que haya más espacio libre.

//Definimos los pines motores que van a controlar la velocidad y la dirección
int dirA = 10;
int dirB = 9;
int speedA = 12;
int speedB = 11;

//Definimos el valor del pin de control sobre el ping del sensor de
//ultrasonidos
const int pingPin = 3;

Servo myservo;  //creamos un objeto servo para controlar el servo motor
            //un máximo de 8 objetos servo se pueden crear

//Ahora iniciamos la parte de configuración
void setup()
{
  Serial.begin(9600);       // establecemos la librería Serie a 9600 bps
  Serial.println("No Me Txoko!");  // escribimos een pantalla ese texto
  pinMode (dirA, OUTPUT);    // ponemos el pin de la direección A como salida
  pinMode (dirB, OUTPUT);    // ponemos el pin de la direección B como salida
  pinMode (speedA, OUTPUT);  // ponemos el pin de la velocidad A como salida
  pinMode (speedB, OUTPUT);  // ponemos el pin de la velocidad B como salida
  myservo.attach(2);  // vinculamos el servo del pin 2 con el objeto servo
}

//Tal y como hemos explicado en la introducción pretendemos que realice varias
//acciones,como son la de moverse con los motores DC, comprobar la distancia
//que existe por delante del robot con el sensor de ultrasonidos continuamente
//Pararse en caso de encontrarse un obstáculo, girar el servo motor a zquierda
//sensor de y derecha para que el untrasonidos compruebe la distancia a cada lado
//, decidir a que lado ir, y por último pararse cada cierto tiempo y realizár
// comprobación de distancia a ambos lados nuevamente.
```

```
void loop()
{

// Envolvemos todo nuestro programa con este bucle for para que se repita cada
//1000 iteraciones para que después realice la comprobación de distancia a
//ambos lados
for(int i=0;i=1000;i++)
{

  long duration, cm;       // definimos la variable que va a estár en centímetros

  // Vamos a lanzar un pulso por el sensor de ultrasonidos para después
  // recogerlo y así poder calcular la distancia que existe entre en sensor y
  // lo que tiene delante (max. 3m)
  pinMode(pingPin, OUTPUT);   // ponemos el pin de control del pulso del
                              // sensor como salida
  digitalWrite(pingPin, LOW);     // mandamos el comienzo de nivel bajo del
                                  // pulso de 2ms
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);    // mandamos la parte central de nivel alto del
                                  // pulso de 5ms
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);     // mandamos lel final de nivel bajo del pulso

  pinMode(pingPin, INPUT);        // ponemos el pin del sensor como entrada

  duration = pulseIn(pingPin, HIGH); // calculamos el tiempo de vuelta
                                     // del pulso

  cm = microsecondsToCentimeters(duration);    // convertimos a distancia
                                               // ese tiempo

  // Mostramos en pantalla el valor obtenido
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  delay(100);

  if (cm < 50)
  {

    // Para los motores DC para realizar la comprobación de distancia a
    // ambos lados
    Serial.println(" parada");
    digitalWrite (speedA, 0);
    digitalWrite (speedB, 0);

    delay(100); // mantiene el motor parado durante 100ms

    // PARTE 1: LADO IZQUIERDO
    // Movemos el servo motor para la izquierda
    Serial.print(" ServoIzquierda");
    myservo.write(145);      // movemos el motor servo hacia la izquierda
    delay(1500);    // esperamos 1500ms para que el servo alcance la posición

    // Ahora vamos a comprobar la distancia que hay a la izquierda del robot
    // y la vamos a guardar de igual manera que hemos explicado anteriormente
    // para tomar distancia con el sensor de ultrasonidos
    long durationLeft, cmLeft;       // definimos la variable que va a estár
```

```
                                    // en centímetros

  pinMode(pingPin, OUTPUT); // ponemos el pin del pulso del sensor
                            // como salida
  digitalWrite(pingPin, LOW);   // mandamos el comienzo de nivel bajo del
                            //pulso de 2ms
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH); // mandamos la parte central de nivel alto
                             // del pulso de 5ms
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);   // mandamos lel final de nivel bajo del pulso

  pinMode(pingPin, INPUT);    // ponemos el pin del sensor como entrada

  durationLeft = pulseIn(pingPin, HIGH);    // calculamos el tiempo de
                                            // vuelta del pulso

  cmLeft = microsecondsToCentimeters(durationLeft); //convertimos a
                                                    // distancia ese tiempo

  // Mostramos en pantalla el valor obtenido
  Serial.print(cmLeft);
  Serial.print("cmLeft");
  Serial.println();

  delay(100);

  // PARTE 2: LADO DERECHO
  // Movemos el servo motor para la derecha
  Serial.print(" ServoDerecha");
  myservo.write(35);      // movemos el motor servo hacia la derecha
  delay(1500);           // esperamos 1500ms para que el servo alcance la
                         // posición

 // Ahora vamos a comprobar la distancia que hay a la derecha del robot y
// la vamos a guardar de igual manera que hemos explicado anteriormente para
// tomar distancia con el sensor de ultrasonidos
  long durationRigth, cmRigth;      // definimos la variable que va a estár
                                    // en centímetros

  pinMode(pingPin, OUTPUT); // ponemos el pin del pulso del sensor
                            // como salida
  digitalWrite(pingPin, LOW);   // mandamos el comienzo de nivel bajo del
                            // pulso de 2ms
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH); // mandamos la parte central de nivel alto
                             // del pulso de 5ms
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);   // mandamos lel final de nivel bajo del pulso

  pinMode(pingPin, INPUT);     // ponemos el pin del sensor como entrada

  durationRigth = pulseIn(pingPin, HIGH);  // calculamos el tiempo de
                                           // vuelta del pulso

  cmRigth = microsecondsToCentimeters(durationRigth); // convertimos a
                                                      // distancia ese tiempo

  // Mostramos en pantalla el valor obtenido
  Serial.print(cmLeft);
```

```
  Serial.print("cmLeft");
  Serial.println();

  delay(100);


// PARTE 3: VOLVER AL CENTRO
  // Movemos el servo motor al centro para que vuelva a poder tomar medidas
  // por delante
  Serial.println(" ServoCentro");
  myservo.write(85);  // movemos el motor servo hacia la posición central
  delay(1500);  // esperamos 1500ms para que el servo alcance la posición

  delay(100);

    // Ahora vamos a decidir según que distancia es la mayor hacia
    // que lado continuar
    // GIRO IZQUIERDA
    if (cmRigth < cmLeft)
    {

      // motor A hacia delante y el B hacia atrás - gira a la izquierda
      Serial.print(" izquierda");
      digitalWrite (dirA, LOW);
      digitalWrite (dirB, HIGH);
          digitalWrite (speedA, 135);
          digitalWrite (speedB, 135);

          delay(500);

    }

    // GIRO DERECHA
    if (cmRigth >= cmLeft)
    {

      // motor A hacia atrás y el B hacia delante - gira a la derecha
      Serial.print(" derecha");
          digitalWrite (dirA, HIGH);
          digitalWrite (dirB, LOW);
          digitalWrite (speedA, 135);
          digitalWrite (speedB, 135);

          delay(500);

    }

}

 if (cm > 50)
 {
   // movemos ambos motores en la misma dirección para que siga
   // para adelante
   Serial.print(" arranca");
   digitalWrite (dirA, HIGH);
   digitalWrite (dirB, HIGH);
   digitalWrite (speedA, 135);
   digitalWrite (speedB, 135);

   delay(100);
```

```
  }

}
// Ahora repetimos la parte del programa que recoge los valores de distancia
// a cada lado del robot, para así poder decidir hacia que lado debe
// continuar. Esta parte es para cada cierto tiempo se pare y mire a ambos
// lados para medir la distancia, y así de esta manera evitar que el robot
// se quede muy cerca de una pared (casi en paralelo) y no la detecte por no
// tener enfrente el obstáculo
// Paramos ambos motores
Serial.println(" parada2");
digitalWrite (speedA, 0);
digitalWrite (speedB, 0);

delay(100); //mantenemos los motores parados durante 100ms antes de continuar

// PARTE 1: LADO IZQUIERDO
// Movemos el servo motor para la izquierda
Serial.print(" ServoIzquierda2");
myservo.write(145);      // movemos el motor servo hacia la izquierda
delay(1500);      // esperamos 1500ms para que el servo alcance la posición

  // Tomamos la distancia a la izquierda del robot
  long durationLeft, cmLeft;

  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);

  pinMode(pingPin, INPUT);
  durationLeft = pulseIn(pingPin, HIGH);

  // Convertimos a distancia
  cmLeft = microsecondsToCentimeters(durationLeft);

  Serial.print(cmLeft);
  Serial.print("cmLeft");
  Serial.println();

  delay(100);

// PARTE 2: LADO DERECHO
// Movemos el servo motor para la derecha
Serial.print(" ServoDerecha2");
myservo.write(35);      // movemos el motor servo hacia la derecha
delay(1500);   // esperamos 1500ms para que el servo alcance la posición


  // Tomamos la distancia a la derecha del robot
  long durationRigth, cmRigth;

  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);
```

```
  pinMode(pingPin, INPUT);
  durationRigth = pulseIn(pingPin, HIGH);

  // Convertimos a distancia
  cmRigth = microsecondsToCentimeters(durationRigth);

  Serial.print(cmRigth);
  Serial.print("cmRigth");
  Serial.println();

  delay(100);
// PARTE 3: VOLVER AL CENTRO
// Movemos el servo motor al centro para que vuelva a poder tomar medidas
// por delante
Serial.println(" ServoCentro2");
myservo.write(85);   // movemos el motor servo hacia la posición central
delay(1500);   // esperamos 1500ms para que el servo alcance la posición

delay(100);

  if (cmRigth < cmLeft)
  {
    // motor A hacia delante y el B hacia atrás - gira a la izquierda
    Serial.print(" izquierda2");
    digitalWrite (dirA, LOW);
    digitalWrite (dirB, HIGH);
    digitalWrite (speedA, 135);
    digitalWrite (speedB, 135);
    delay(500);
  }

  if (cmRigth >= cmLeft)
  {

    // motor A hacia atrás y el B hacia delante - gira a la derecha
    Serial.print(" derecha2");
    digitalWrite (dirA, HIGH);
    digitalWrite (dirB, LOW);
    digitalWrite (speedA, 135);
    digitalWrite (speedB, 135);
    delay(500);

  }

// movemos ambos motores en la misma dirección para que siga para adelante
Serial.print(" arranca2");
digitalWrite (dirA, HIGH);
digitalWrite (dirB, HIGH);
digitalWrite (speedA, 135);
digitalWrite (speedB, 135);
delay (100); // mantener los motores girando durante 100ms

}

//Función que convierte un valor en microsegundos a un valor en centímetros
long microsecondsToCentimeters(long microseconds)
{
  return microseconds / 29 / 2;
}
```

Robot 1: No Me Txoko Mejorado UPNA

```
/* NoMeTxoko Mejorado UPNA */

//Este ejemplo es un programa para arduino que utiliza la librería de arduino que aparece
//a continuación, y que está diseñada para la utilización de un servo motor. Además utiliza
//un sensor de ultrasonidos PING)))™ Ultrasonic Distance Sensor (#28015).

#include <Servo.h>

//En esta ocasión utilizamos la placa Arduino AT168 Duemilanove la cual es suficiente para
//lo que necesitamos; es decir para el uso de dos motores DC, un servo motor y un sensor de
//ultrasonidos.

//Nuestra intención es que el robot se desplace por la estancia sin ningún recorrido marcado
//sin chocarse con ningún elemento de la misma:

//El primer bloque del programa va a consistir en un bucle a modo de contador en el que
//está contenido todo el giro de l servo motor hacia la izquierda, y que cada 45º compruebe
//la distancia que existe en la dirección a la que apuna en sensor y decida en que dirección
//seguir.
//Para la segunda fase del programa vamos a relizar el mismo sistema pero con el movimiento
//del servo motor hacia la derecha. Tomando las medidas cada 45º también.

//Definimos los pines motores que van a controlar la velocidad y la dirección
int dirA = 10;
int dirB = 9;
int speedA = 12;
int speedB = 11;

//Definimos el valor del pin de control sobre el ping del sensor de ultrasonidos
const int pingPin = 3;

Servo myservo;  //creamos un objeto servo para controlar el servo motor
            //un máximo de 8 objetos servo se pueden crear

int a = 1;      //Contadores
int a0 = 0;
int a45A = 0;
int a45B = 0;
int a90A = 0;
int a90B = 0;
int a135A = 0;
int a135B = 0;
int a180 = 0;

//Ahora iniciamos la parte de configuración
void setup()
{

  Serial.begin(9600);          // establecemos la librería Serie a 9600 bps
  Serial.println("No Me Txoko!");  // escribimos een pantalla ese texto
  pinMode (dirA, OUTPUT);       // ponemos el pin de la direección A como salida
  pinMode (dirB, OUTPUT);       // ponemos el pin de la direección B como salida
  pinMode (speedA, OUTPUT);  // ponemos el pin de la velocidad A como salida
  pinMode (speedB, OUTPUT);  // ponemos el pin de la velocidad B como salida
  myservo.attach(2);  // vinculamos el servo del pin 2 con el objeto servo
}
```

```
//Tal y como hemos explicado en la introducción pretendemos que realice varias acciones,
//como son la de moverse con los motores DC, comprobar la distancia que existe por delante
//del robot con el sensor de ultrasonidos continuamente, pararse en caso de encontrarse un
//obstáculo, girar el servo motor a zquierda y derecha para que el sensor de untrasonidos
//compruebe la distancia a cada lado, decidir a que lado ir, y por último pararse cada cierto
//tiempo y realizár comprobación de distancia a ambos lados nuevamente.
void loop()
{

  // Con este bucle for pretendemos recorrer un arco de 0º a 180º en el que vamos a tomar
  // medidas con el sensor cada 45º, para después realizar la comprobación de distancia <50cm.

  if (a==1){
   // Movemos ambos motores en la misma dirección para que siga para adelante
   Serial.print(" Arranca ");
    digitalWrite (dirA, HIGH);
    digitalWrite (dirB, HIGH);
    analogWrite (speedA, 250);
    analogWrite (speedB, 250);
   a=0;
  }
                  ////////////////////////////////////
                  // PASO 1: GIRO DEL SERVO IZQUIERDA //

  // GRADOS: 0º ////////////////////////////////////////////////////////////////////////
  // Movemos el servo motor
  Serial.print(" Mover Servo a 0º ");
  myservo.write(10);   // movemos servo motor a 10º para que no se bloquee en el límite de 0º
  delay (350);

  long duration, cm;                  // definimos la variable que va a estár en centímetros

  // Vamos a lanzar un pulso por el sensor de ultrasonidos para después recogerlo y así poder
  // calcular la distancia que existe entre en sensor y lo que tiene delante (max. 3m)
  pinMode(pingPin, OUTPUT);   // ponemos el pin de control del pulso del sensor como salida
  digitalWrite(pingPin, LOW);     // mandamos el comienzo de nivel bajo del pulso de 2ms
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);  // mandamos la parte central de nivel alto del pulso de 5ms
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);     // mandamos lel final de nivel bajo del pulso

  pinMode(pingPin, INPUT);       // ponemos el pin del sensor como entrada

  duration = pulseIn(pingPin, HIGH);       // calculamos el tiempo de vuelta del pulso
  cm = microsecondsToCentimeters(duration);    // convertimos a distancia ese tiempo

  // Mostramos en pantalla el valor obtenido
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  if (a0==1){
   // Movemos ambos motores en la misma dirección para que siga para adelante
   Serial.print(" Arranca ");
    digitalWrite (dirA, HIGH);
    digitalWrite (dirB, HIGH);
    analogWrite (speedA, 250);
    analogWrite (speedB, 250);
    a0=0;
  }
```

```
// En el caso de que la distancia a un obstáculo sea inferior a 50cm el robot girará
if (cm < 50)
{
  // motor A hacia delante y el B hacia atrás - gira a la izquierda
  Serial.print(" Giro a 0º ");
  digitalWrite (dirA, LOW);
  digitalWrite (dirB, HIGH);
  analogWrite (speedA, 250);
  analogWrite (speedB, 250);
  delay(650);
  a0=1;

  // Paramos ambos motores para que continue la comprobación
  Serial.print(" Parada ");
  digitalWrite (dirA, LOW);
  digitalWrite (dirB, LOW);
  analogWrite (speedA, 0);
  analogWrite (speedB, 0);
}

// GRADOS: 45º ////////////////////////////////////////////////////////////////////////////
// Movemos el servo motor
Serial.print(" Mover Servo a 45º ");
myservo.write(40);    // movemos el motor servo a 40º para que quede ajustado todo el giro
delay (350);

// Vamos a lanzar un pulso por el sensor de ultrasonidos para después recogerlo y así poder
// calcular la distancia que existe entre en sensor y lo que tiene delante (max. 3m)
pinMode(pingPin, OUTPUT);   // ponemos el pin de control del pulso del sensor como salida
digitalWrite(pingPin, LOW);     // mandamos el comienzo de nivel bajo del pulso de 2ms
delayMicroseconds(2);
digitalWrite(pingPin, HIGH);  // mandamos la parte central de nivel alto del pulso de 5ms
delayMicroseconds(5);
digitalWrite(pingPin, LOW);     // mandamos lel final de nivel bajo del pulso

pinMode(pingPin, INPUT);       // ponemos el pin del sensor como entrada

duration = pulseIn(pingPin, HIGH);      // calculamos el tiempo de vuelta del pulso
cm = microsecondsToCentimeters(duration);    // convertimos a distancia ese tiempo

// Mostramos en pantalla el valor obtenido
Serial.print(cm);
Serial.print("cm");
Serial.println();

if (a45A==1){
 // Movemos ambos motores en la misma dirección para que siga para adelante
 Serial.print(" Arranca ");
  digitalWrite (dirA, HIGH);
  digitalWrite (dirB, HIGH);
  analogWrite (speedA, 250);
  analogWrite (speedB, 250);
  a45A=0;
}

// En el caso de que la distancia a un obstáculo sea inferior a 50cm el robot girará
if (cm < 50)
{
  // motor A hacia delante con v1 y el B delante con v2 - giro a la izquierda suave
```

```
    Serial.print(" Giro a 45º ");
    digitalWrite (dirA, HIGH);
    digitalWrite (dirB, HIGH);
    analogWrite (speedA, 125);
    analogWrite (speedB, 250);
    delay(650);
    a45A=1;

// Paramos ambos motores para que continue la comprobación
    Serial.print(" Parada ");
    digitalWrite (dirA, LOW);
    digitalWrite (dirB, LOW);
    analogWrite (speedA, 0);
    analogWrite (speedB, 0);
  }

  // GRADOS: 90º ////////////////////////////////////////////////////////////////////////
  // Movemos el servo motor
  Serial.print(" Mover Servo a 90º ");
  myservo.write(85);      // movemos el motor servo s 85º para ajustar todo el giro
  delay (350);

  // Vamos a lanzar un pulso por el sensor de ultrasonidos para después recogerlo y así poder
  // calcular la distancia que existe entre en sensor y lo que tiene delante (max. 3m)
  pinMode(pingPin, OUTPUT);   // ponemos el pin de control del pulso del sensor como salida
  digitalWrite(pingPin, LOW);     // mandamos el comienzo de nivel bajo del pulso de 2ms
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);  // mandamos la parte central de nivel alto del pulso de 5ms
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);     // mandamos lel final de nivel bajo del pulso

  pinMode(pingPin, INPUT);      // ponemos el pin del sensor como entrada

  duration = pulseIn(pingPin, HIGH);      // calculamos el tiempo de vuelta del pulso
  cm = microsecondsToCentimeters(duration);    // convertimos a distancia ese tiempo

  // Mostramos en pantalla el valor obtenido
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  if (a90A==1){
   // Movemos ambos motores en la misma dirección para que siga para adelante
   Serial.print(" Arranca ");
    digitalWrite (dirA, HIGH);
    digitalWrite (dirB, HIGH);
    analogWrite (speedA, 250);
    analogWrite (speedB, 250);
    a90A=0;
  }

  // En el caso de que la distancia a un obstáculo sea inferior a 50cm el robot girará
  if (cm < 50)
  {
    // motor A hacia delante y el B hacia atrás - gira a la izquierda
    Serial.print(" Giro a 90º ");
    digitalWrite (dirA, LOW);
    digitalWrite (dirB, HIGH);
    analogWrite (speedA, 250);
    analogWrite (speedB, 250);
```

```
  delay(650);
  a90A=1;

  // Paramos ambos motores para que continue la comprobación
  Serial.print(" Parada ");
  digitalWrite (dirA, LOW);
  digitalWrite (dirB, LOW);
  analogWrite (speedA, 0);
  analogWrite (speedB, 0);
}

// GRADOS: 135º ////////////////////////////////////////////////////////////////////////////
// Movemos el servo motor
Serial.print(" Mover Servo a 135º ");
myservo.write(130);      // movemos el motor servo a 130º para ajustar todo el giro
delay (350);

// Vamos a lanzar un pulso por el sensor de ultrasonidos para después recogerlo y así poder
// calcular la distancia que existe entre en sensor y lo que tiene delante (max. 3m)
pinMode(pingPin, OUTPUT);   // ponemos el pin de control del pulso del sensor como salida
digitalWrite(pingPin, LOW);      // mandamos el comienzo de nivel bajo del pulso de 2ms
delayMicroseconds(2);
digitalWrite(pingPin, HIGH);  // mandamos la parte central de nivel alto del pulso de 5ms
delayMicroseconds(5);
digitalWrite(pingPin, LOW);      // mandamos lel final de nivel bajo del pulso

pinMode(pingPin, INPUT);       // ponemos el pin del sensor como entrada

duration = pulseIn(pingPin, HIGH);       // calculamos el tiempo de vuelta del pulso
cm = microsecondsToCentimeters(duration);    // convertimos a distancia ese tiempo

// Mostramos en pantalla el valor obtenido
Serial.print(cm);
Serial.print("cm");
Serial.println();

if (a135A==1){
 // Movemos ambos motores en la misma dirección para que siga para adelante
 Serial.print(" Arranca ");
  digitalWrite (dirA, HIGH);
  digitalWrite (dirB, HIGH);
  analogWrite (speedA, 250);
  analogWrite (speedB, 250);
  a135A=0;
}

// En el caso de que la distancia a un obstáculo sea inferior a 50cm el robot girará
if (cm < 50)
{
  // motor A hacia delante con v1 y el B hacia delante con v2 - giro a la derecha suave
  Serial.print(" Giro a 135º ");
  digitalWrite (dirA, HIGH);
  digitalWrite (dirB, HIGH);
  analogWrite (speedA, 250);
  analogWrite (speedB, 125);
  delay(650);
  a135A=1;

  // Paramos ambos motores para que continue la comprobación
  Serial.print(" Parada ");
```

```
  digitalWrite (dirA, LOW);
  digitalWrite (dirB, LOW);
  analogWrite (speedA, 0);
  analogWrite (speedB, 0);
}

// GRADOS: 180º //////////////////////////////////////////////////////////////////////
// Movemos el servo motor
Serial.print(" Mover Servo a 180º ");
myservo.write(165);//movemos motor servo a 165º para que no se bloquee en el límite de 180º
delay (350);

// Vamos a lanzar un pulso por el sensor de ultrasonidos para después recogerlo y así poder
// calcular la distancia que existe entre en sensor y lo que tiene delante (max. 3m)
pinMode(pingPin, OUTPUT);   // ponemos el pin de control del pulso del sensor como salida
digitalWrite(pingPin, LOW);     // mandamos el comienzo de nivel bajo del pulso de 2ms
delayMicroseconds(2);
digitalWrite(pingPin, HIGH);  // mandamos la parte central de nivel alto del pulso de 5ms
delayMicroseconds(5);
digitalWrite(pingPin, LOW);     // mandamos lel final de nivel bajo del pulso

pinMode(pingPin, INPUT);      // ponemos el pin del sensor como entrada

duration = pulseIn(pingPin, HIGH);      // calculamos el tiempo de vuelta del pulso
cm = microsecondsToCentimeters(duration);    // convertimos a distancia ese tiempo

// Mostramos en pantalla el valor obtenido
Serial.print(cm);
Serial.print("cm");
Serial.println();

if (a180==1){
 // Movemos ambos motores en la misma dirección para que siga para adelante
 Serial.print(" Arranca ");
  digitalWrite (dirA, HIGH);
  digitalWrite (dirB, HIGH);
  analogWrite (speedA, 250);
  analogWrite (speedB, 250);
  a180=0;
}

// En el caso de que la distancia a un obstáculo sea inferior a 50cm el robot girará
if (cm < 50)
{
  // motor A hacia atrás y el B hacia delante - gira a la derecha
  Serial.print(" Giro a 180º ");
  digitalWrite (dirA, HIGH);
  digitalWrite (dirB, LOW);
  analogWrite (speedA, 250);
  analogWrite (speedB, 250);
  delay(650);
  a180=1;

  // Paramos ambos motores para que continue la comprobación
  Serial.print(" Parada ");
  digitalWrite (dirA, LOW);
  digitalWrite (dirB, LOW);
  analogWrite (speedA, 0);
  analogWrite (speedB, 0);
}
```

```
                ////////////////////////////////////
                // PASO 2: GIRO DEL SERVO DERECHA //

// GRADOS: 135º ////////////////////////////////////////////////////////////////////////
// Movemos el servo motor
Serial.print(" Mover Servo a 135º ");
myservo.write(130);   // movemos el motor servo a 130º para que quede ajustado todo el giro
delay (350);

// Vamos a lanzar un pulso por el sensor de ultrasonidos para después recogerlo y así poder
// calcular la distancia que existe entre en sensor y lo que tiene delante (max. 3m)
pinMode(pingPin, OUTPUT);   // ponemos el pin de control del pulso del sensor como salida
digitalWrite(pingPin, LOW);     // mandamos el comienzo de nivel bajo del pulso de 2ms
delayMicroseconds(2);
digitalWrite(pingPin, HIGH);  // mandamos la parte central de nivel alto del pulso de 5ms
delayMicroseconds(5);
digitalWrite(pingPin, LOW);     // mandamos lel final de nivel bajo del pulso

pinMode(pingPin, INPUT);       // ponemos el pin del sensor como entrada

duration = pulseIn(pingPin, HIGH);      // calculamos el tiempo de vuelta del pulso
cm = microsecondsToCentimeters(duration);    // convertimos a distancia ese tiempo

// Mostramos en pantalla el valor obtenido
Serial.print(cm);
Serial.print("cm");
Serial.println();

if (a135B==1){
 // Movemos ambos motores en la misma dirección para que siga para adelante
 Serial.print(" Arranca ");
  digitalWrite (dirA, HIGH);
  digitalWrite (dirB, HIGH);
  analogWrite (speedA, 250);
  analogWrite (speedB, 250);
  a135B=0;
}

// En el caso de que la distancia a un obstáculo sea inferior a 50cm el robot girará
if (cm < 50)
{
  // motor A hacia delante con v1 y el B hacia delante con v2 - giro a la derecha suave
  Serial.print(" Giro a 135º ");
  digitalWrite (dirA, HIGH);
  digitalWrite (dirB, HIGH);
  analogWrite (speedA, 250);
  analogWrite (speedB, 125);
  delay(650);
  a135B=1;

  // Paramos ambos motores para que continue la comprobación
  Serial.print(" Parada ");
  digitalWrite (dirA, LOW);
  digitalWrite (dirB, LOW);
  analogWrite (speedA, 0);
  analogWrite (speedB, 0);
}

// GRADOS: 90º ////////////////////////////////////////////////////////////////////////
// Movemos el servo motor
```

```
Serial.print(" Mover Servo a 90º ");
myservo.write(85);  // movemos el motor servo a 85º para que quede ajustado todo el giro
delay (350);

// Vamos a lanzar un pulso por el sensor de ultrasonidos para después recogerlo y así poder
// calcular la distancia que existe entre en sensor y lo que tiene delante (max. 3m)
pinMode(pingPin, OUTPUT);   // ponemos el pin de control del pulso del sensor como salida
digitalWrite(pingPin, LOW);     // mandamos el comienzo de nivel bajo del pulso de 2ms
delayMicroseconds(2);
digitalWrite(pingPin, HIGH);  // mandamos la parte central de nivel alto del pulso de 5ms
delayMicroseconds(5);
digitalWrite(pingPin, LOW);     // mandamos lel final de nivel bajo del pulso

pinMode(pingPin, INPUT);      // ponemos el pin del sensor como entrada

duration = pulseIn(pingPin, HIGH);      // calculamos el tiempo de vuelta del pulso
cm = microsecondsToCentimeters(duration);    // convertimos a distancia ese tiempo

// Mostramos en pantalla el valor obtenido
Serial.print(cm);
Serial.print("cm");
Serial.println();

if (a90B==1){
 // Movemos ambos motores en la misma dirección para que siga para adelante
 Serial.print(" Arranca ");
  digitalWrite (dirA, HIGH);
  digitalWrite (dirB, HIGH);
  analogWrite (speedA, 250);
  analogWrite (speedB, 250);
  a90B=0;
}

// En el caso de que la distancia a un obstáculo sea inferior a 50cm el robot girará
if (cm < 50)
{
  // motor A hacia atrás y el B hacia adelante - gira a la derecha
  Serial.print(" Giro a 90º ");
  digitalWrite (dirA, HIGH);
  digitalWrite (dirB, LOW);
  analogWrite (speedA, 250);
  analogWrite (speedB, 250);
  delay(650);
  a90B=1;

  // Paramos ambos motores para que continue la comprobación
  Serial.print(" Parada ");
  digitalWrite (dirA, LOW);
  digitalWrite (dirB, LOW);
  analogWrite (speedA, 0);
  analogWrite (speedB, 0);
}

// GRADOS: 45º ////////////////////////////////////////////////////////////////////////////
// Movemos el servo motor
Serial.print(" Mover Servo a 45º ");
myservo.write(40);   // movemos el motor servo a 40º para que quede ajustado todo el giro
delay (350);

// Vamos a lanzar un pulso por el sensor de ultrasonidos para después recogerlo y así poder
```

```
  // calcular la distancia que existe entre en sensor y lo que tiene delante (max. 3m)
  pinMode(pingPin, OUTPUT);   // ponemos el pin de control del pulso del sensor como salida
  digitalWrite(pingPin, LOW);     // mandamos el comienzo de nivel bajo del pulso de 2ms
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);  // mandamos la parte central de nivel alto del pulso de 5ms
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);     // mandamos lel final de nivel bajo del pulso

  pinMode(pingPin, INPUT);      // ponemos el pin del sensor como entrada

  duration = pulseIn(pingPin, HIGH);      // calculamos el tiempo de vuelta del pulso
  cm = microsecondsToCentimeters(duration);    // convertimos a distancia ese tiempo

  // Mostramos en pantalla el valor obtenido
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  if (a45B==1){
   // Movemos ambos motores en la misma dirección para que siga para adelante
   Serial.print(" Arranca ");
    digitalWrite (dirA, HIGH);
    digitalWrite (dirB, HIGH);
    analogWrite (speedA, 250);
    analogWrite (speedB, 250);
    a45B=0;
  }

  // En el caso de que la distancia a un obstáculo sea inferior a 50cm el robot girará
  if (cm < 50)
  {
    // motor A hacia delante con v1 y el B delante con v2 - giro a la izquierda suave
    Serial.print(" Giro a 45º ");
    digitalWrite (dirA, HIGH);
    digitalWrite (dirB, HIGH);
    analogWrite (speedA, 125);
    analogWrite (speedB, 250);
    delay(650);
    a45B=1;

    // Paramos ambos motores para que continue la comprobación
    Serial.print(" Parada ");
    digitalWrite (dirA, LOW);
    digitalWrite (dirB, LOW);
    analogWrite (speedA, 0);
    analogWrite (speedB, 0);
  }

                    ////////////////////////////////////
                    // FIN DE BUCLE MOVIMIENTO SERVO //
                    ////////////////////////////////////
}

//Función que convierte un valor en microsegundos a un valor en centímetros
long microsecondsToCentimeters(long microseconds)
{
  return microseconds / 29 / 2;
}
```

Robot 2. Seguidor de Líneas

```
/* SEGUIDOR DE LINEA UPNA */

//Este ejemplo es un programa para arduino que utiliza la librería de Pololu que
//aparece a continuación, y que está diseñada para seguidores de línea y
//sensores QTR-1RC y QTR-8RC.

#include <PololuQTRSensors.h>

//Este ejemplo se ha diseñado para su uso con ocho sensores de QTR-1RC o un
//módulo de ocho sensores de QTR-8RC. Estos sensores de reflectancia se debe
//conectar a las entradas digitales 3 a 10, aubque en este ejemplo como solo
//utilizamos 6 sensores y el pin de control del emisor no lo activamos
//conectaremos a los pines 2 a 7. El pin de control del emisor, opcionalmente se
//puede conectar al pin digital de 2, o puede dejarse que desconectado y cambiar
//el EMITTER_PIN # definido por QTR_NO_EMITTER_PIN.

//En la fase de instalación de este ejemplo se calibra el sensor durante diez
//segundos y se enciende un LED de control, mientras que la calibración que está
//produciendo. Durante esta fase, se debe exponer cada sensor de reflectancia a
//las lecturas más claras y más oscuras con las que se vaya ha encontrar. Por
//ejemplo, si usted está haciendo un seguidor de línea, usted debe deslizarse a
//través de los sensores de la línea durante la fase de calibrado de forma que
//cada sensor puede obtener una lectura de cuanto oscura es la linea y cun claro
//es el suelo. Una calibración incorrecta dará lugar a lecturas pobres.
//Si se desea omitir la fase de calibración, se puede obtener las lecturas del
//sensor (tiempos de pulso desde 0 hasta 2500 us) llamando a la función
//qtra.read (sensorValues) en lugar de qtra.readLine (sensorValues).

//El bucle principal del ejemplo lee los valores de los sensores calibrados y
//los utiliza para estimar la posición de una línea. Se puede probar esto con un
//pedazo de cinta adhesiva  de 3 / 4 "cinta aislante negra pegada sobre un
//pedazo de papel blanco y deslizar el sensor a través de ella. Imprime los
//valores del sensor al monitor en serie, según los números del 0 (máxima
//reflectancia) a 9 (mínimo reflectancia), seguido de la ubicación aproximada de
//la línea como un número del 0 al 5000. 1000: la línea está directamente debajo
//del sensor 1, 2000 significa directamente debajo del sensor 2, etc. 0
//significa que la línea está directamente debajo del sensor 0 o fue visto por
//ultima vez por el sensor 0 antes de que se pierdan. 5000: la línea está
//directamente debajo del sensor de 5 o fue visto por última vez por el sensor
//de 5 antes de que se pierdan.

//Definimos el valor de varias constantes que utiliza el programa
#define NUM_SENSORS   6     // número de sensores utilizados
#define TIMEOUT       2500  // espera 2500us para apagar los sensores
#define QTR_NO_EMITTER_PIN  // el emisor no está controlado

//Los sensores del 1 al 6 están conectados a los pines digitales 2 a 7
PololuQTRSensorsRC qtrrc((unsigned char[]) {2, 3, 4, 5, 6, 7},  NUM_SENSORS, TIMEOUT);
unsigned int sensorValues[NUM_SENSORS];  //inicializamos una variable vacía

//Los pines de control de los motores van a ser 10 y 11 para la dirección,
//y 12 y 13 para la velocidad. Los pines 10 y 12 serán de un motro y 11 y 13
//del otro.
int dirA = 11;
int dirB = 10;
int speedA = 13;
int speedB = 12;
```

```
//Comienza la fase de instalación en la que se va ha realizar la calibración
void setup()
{

  // definimos los pines de control de los motores como salida
  pinMode (dirA, OUTPUT);
  pinMode (dirB, OUTPUT);
  pinMode (speedA, OUTPUT);
  pinMode (speedB, OUTPUT);

  delay(500);                    // comenzamos con 500ms de espera

  int i;                  // inicilizamos el contador i
  pinMode(13, OUTPUT);              // ponemos el pin del LED como salida
  digitalWrite(13, HIGH);   // encendemos el LED para iniciar la calibracion

  for (i = 0; i < 400; i++) // hace que la calibración dure unos 10 segundos
  {
    qtrrc.calibrate();      // todos los sensores realizan 10 lecturas de 2500us
  }
  digitalWrite(13, LOW);   // apagamos el LED para acabar el modo de calibracion

  //escribe en pantalla la media de los mínimos valores medidos para cada sensor
  Serial.begin(9600);  // velocidad de transmisión entre la placa y el ordenador
  for (i = 0; i < NUM_SENSORS; i++)
  {
    Serial.print(qtrrc.calibratedMinimumOn[i]);
    Serial.print(' ');
  }
  Serial.println();

  //escribe en pantalla la media de los máximos valores medidos para cada sensor
  for (i = 0; i < NUM_SENSORS; i++)
  {
    Serial.print(qtrrc.calibratedMaximumOn[i]);
    Serial.print(' ');
  }

  Serial.println();
  Serial.println();
  delay(1000);

}

//Una vez realizada la calibración comienza el bucle principal donde pretende
//que sigua una línea negra sobre un fondo blanco

void loop()
{
  // lee los valores de calibrado de los sensores y obtiene una medida de la
  // posición de línea 0 a 5000, donde 0 significa que directamente debajo del
  // sensor 0 o la línea se perdió pasado el sensor 0, 1000 significa que
  // directamente debajo del sensor 1, 2000 significa que directamente debajo
  // del sensor 2, etc.
  // Nota: los valores devueltos serán incorrectos si los sensores no han sido
  // adecuadamente calibrados durante la fase de calibración.  Para obtener los
  // valores en bruto del sensor llamar a la función: qtra.read(sensorValues);

  unsigned int position = qtrrc.readLine(sensorValues);
```

```
//qtrrc.read(sensorValues);  // en caso de querer leer los valores en bruto

// escribir los valores de los sensores como un número del 0 al 9, donde 0
// significa máxima reflectancia y 9 mínima reflectancia, seguido de la
// posición de la línea
unsigned char i;
for (i = 0; i < NUM_SENSORS; i++)
{
  Serial.print(sensorValues[i] * 10 / 1001);
  Serial.print(' ');
}
Serial.print("    ");
Serial.println(position);

// Una vez conocemos la posición de la línea con respecto a los sensores
// podemos decidir que acciones debe realiazar para ir siguiendo la línea

// En caso de que la línea se encuentre entre los sensores 2 y 3 queremos que
// siga hacia adelante, de manera que le damos la misma dirección y velocidad
// a ambos motores
if (position > 2400&&position < 2600)
{
  Serial.print(" arranca");
  analogWrite (dirA, 255);
  analogWrite (dirB, 255);
  digitalWrite (speedA, LOW);
  digitalWrite (speedB, LOW);
}

// En caso de que la línea se encuentre a la derecha del sensor 2 queremos
// que gire hacia la izquierda y vuelva a la línea, así que le damos la misma
// dirección a los dos motores pero velocidades distintas. Le damos menos
// velocidad a la rueda interior del giro
if (position < 2400)
{
  Serial.print(" izquierda");
  analogWrite (dirA, 255);
  analogWrite (dirB, 50);
  digitalWrite (speedA, LOW);
  digitalWrite (speedB, LOW);
}

// En caso de que la línea se encuentre a la izquierda del sensor 3 queremos
// que gire hacia la derecha y vuelva a la línea, así que le damos la misma
// dirección a los dos motores pero velocidades distintas. Le damos menos
// velocidad a la rueda interior del giro
if (position > 2600)
{
   Serial.println(" derecha");
   analogWrite (dirA, 50);
   analogWrite (dirB, 255);
   digitalWrite (speedA, LOW);
   digitalWrite (speedB, LOW);
}
}
```

Robot 3: Bee-Bot Arduino UPNA

```
/* Bee-Bot UPNA */

//Este ejemplo es un programa para arduino que utiliza las librerías de arduino
//que aparecen a continuación, y que están diseñadas para la utilización de una
//pantalla LCD de 20x4 y un teclado matricial de 12 teclas.

#include <LiquidCrystal.h>
#include <Keypad.h>

//Para este caso utilizamos la placa Arduino ATMega que tiene más pines de
//conexión ya que necesitamos un número mayor para controlar la pantalla LCD, el
//tecladdo numérico y los motores DC.

//Pretendemos que nuestro robot realice una serie de movimientos que se le abrán
//indicado en el teclado matricial para después realizarlos todos seguidos. Tras
//asignar todos los pines de los componentes a los pines de la placa vamos a
//separa el programa en dos parte.

//La primera parte va a constar en la fase de recogida de datos, es decir que
//cuando se vayan introduciendo los valores de direcciones (2, 4, 6 y 8) se
//vayan guardando, y cuando se pulse el bot´pon de START (5) comience el
//programa. También hemos incluido las acciones de CLEAR (7) y la de mostrar el
//vector de movimiento (9).

//En la segunda fase del programa va realizando la lectura del vector de
//movimientos y va indicando a los motores los movimiento que tiene que
//realizar.

//Los pines de conexión de la pantalla LCD con la placa Arduino son los valores
//pares de los pines 38 a 48
LiquidCrystal lcd( 38, 40, 42, 44, 46, 48);

//Inciciamos una serie de variables
int vector[10]={0};    //definivos el número de entradas de nuestro vector
int i=0;         //inicializamos el puntero i
int k=0;         //inicializamos el puntero k
int N=10; //inicializamos una constante que el tamaño del vector de movimientos

const byte ROWS = 4;   //definimos que nuestro teclado va a tener cuatro filas
const byte COLS = 3;   //definimos que nuestro teclado va a tener tres columnas

//Definimos el KEYMAP,es decir cual es la posición matricial de nuestro teclado
char keys[ROWS][COLS] = {
  {1,2,3},
  {4,5,6},
  {7,8,9}
};

//Conexión del keypad ROW0, ROW1, ROW2 y ROW3 a estos pins de Arduino.
byte rowPins[ROWS] = { 41, 51, 49, 45 };
//Conexión del keypad COL0, COL1 and COL2 a estos pins de Arduino.
byte colPins[COLS] = { 43, 39, 47 };

//Creamos el Keypad llamando a la siguente función y con los valores ante
//asignados
Keypad kpd = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

//Definimos los pines motores que van a controlar la velocidad y la dirección
```

```
int dirA = 11;
int dirB = 10;
int speedA = 13;
int speedB = 12;

//Definimos los pines de los LEDs que nos van a ofrecer información del estado
//del programa
int LED1 = 8;
int LED2 = 9;

// Ahora iniciamos la parte de configuración

void setup()
{
  Serial.begin(9600); // Inicializa la comunicacion serie
  lcd.begin(20, 4);  // Definimos el número de filas y columnas de la pantalla
  pinMode (dirA, OUTPUT);     // Ponemos el pin de la direección A como salida
  pinMode (dirB, OUTPUT);     // Ponemos el pin de la direección B como salida
  pinMode (speedA, OUTPUT);   // Ponemos el pin de la velocidad A como salida
  pinMode (speedB, OUTPUT);   // Ponemos el pin de la velocidad B como salida
  pinMode (LED1, OUTPUT);     // Ponemos el pin del LED1 como salida
  pinMode (LED2, OUTPUT);     // Ponemos el pin del LED2 como salida
}

//Comienza el programa que va a separarse en dos fases, la fase de escritura
//(pulsación más grabación de datos) que incluye todos los casos del bucle
//principal switch; y la fase de lectura que comienza y acaba en el caso 5 y
//que es donde realiza todos los movimentos previamente definidos en la parte
//de escritura.

void loop() {

Serial.println("Comienza");
lcd.setCursor(0, 0);       // Apuntamos a la posición (1,1) de la pantalla
lcd.print("BEE-BOT ARDUINO UPNA");   // Saca esa frase en pantalla


   // parte de lectura de datos del keyboard (teclado)
  int key = kpd.getKey();

  if(key)  // lo mismo que if(key != NO_KEY)
  {
   switch (key)
    {
      //Caso de pulsación hacia ARRIBA
      case 2:
        vector[i]=key;         // Introducimos el valor de la pulsación en la posición
                               // i del vector
        i++;     // Sumamos 1 al puntero para que apunte a la siguiente posición
                 // del vector
      break;

      //Caso de pulsación hacia la IZQUIERDA
      case 4:
        vector[i]=key;
        i++;
      break;

      //Caso de pulsación hacia la DERECHA
      case 6:
```

```
      vector[i]=key;
      i++;
    break;

//Caso de pulsación hacia ATRÁS
    case 8:
      vector[i]=key;
      i++;
    break;

    //Caso de pulsación para dar comienzo al movimiento
    case 5:

      Serial.println("START");     // Mostramos la palabra comienzo
       digitalWrite(LED1, HIGH);      // Encendemos el LED1 para comenzar
                                              // el movimiento
      lcd.setCursor(7, 1);   //Ponemos el puntero en la 2ª línea
      lcd.print("START ");
       delay(100);
       digitalWrite(LED1, LOW);
       k=0;          // Inicializamos el contador del siguente bucle para leer
                     // el vector

      do {

       switch (vector[k]) {
         case 2:    // Realiza un paso hacia adelelante
           Serial.println("adelante");
            digitalWrite (dirA, HIGH);
            digitalWrite (dirB, HIGH);
            analogWrite (speedA, 245);
            analogWrite (speedB, 250);
               lcd.setCursor(7, 1);
           lcd.print("  UP  ");
           delay(400); // realiza el movimiento durante 1 segundo
           Serial.println("parada");
            digitalWrite (dirA, LOW);
            digitalWrite (dirB, LOW);
           delay(300); // realiza el movimiento durante 300 milisegundos
           k++;
           break;

         case 4:    // Realiza un paso hacia la izquierda
           Serial.println("izquierdaPaso1");
            digitalWrite (dirA, HIGH);
            digitalWrite (dirB, HIGH);
            analogWrite (speedA, 245);
            analogWrite (speedB, 10);
               lcd.setCursor(7, 1);
           lcd.print(" LEFT ");
           delay(385); // realiza el movimiento durante 1 segundo
           Serial.println("izquierdaPaso2");
            digitalWrite (dirA, HIGH);
            digitalWrite (dirB, HIGH);
            analogWrite (speedA, 245);
            analogWrite (speedB, 250);
           delay(400); // realiza el movimiento durante 1 segundo
           Serial.println("parada");
            digitalWrite (dirA, LOW);
            digitalWrite (dirB, LOW);
```

```
      delay(300); // realiza el movimiento durante 300 milisegundos
      k++;
      break;



    case 6:    // Realiza un paso hacia la derecha
      Serial.println("derechaPaso1");
       digitalWrite (dirA, HIGH);
       digitalWrite (dirB, HIGH);
       analogWrite (speedA, 5);
       analogWrite (speedB, 250);
          lcd.setCursor(7, 1);
      lcd.print("RIGHT ");
      delay(400); // realiza el movimiento durante 1 segundo
      Serial.println("derechaPaso2");
       digitalWrite (dirA, HIGH);
       digitalWrite (dirB, HIGH);
       analogWrite (speedA, 245);
       analogWrite (speedB, 250);
      delay(400); // realiza el movimiento durante 1 segundo
      Serial.println("parada");
       digitalWrite (dirA, LOW);
       digitalWrite (dirB, LOW);
      delay(300); // realiza el movimiento durante 300 milisegundos
      k++;
      break;

    case 8:    // Realiza un paso hacia atras
          lcd.setCursor(7, 1);
      lcd.print(" DOWN ");
      Serial.println("atrasPaso1");
       digitalWrite (dirA, HIGH);
       digitalWrite (dirB, HIGH);
       analogWrite (speedA, 5);
       analogWrite (speedB, 250);
      delay(755); // realiza el movimiento durante 1 segundo
      Serial.println("atrasPaso2");
       digitalWrite (dirA, HIGH);
       digitalWrite (dirB, HIGH);
       analogWrite (speedA, 245);
       analogWrite (speedB, 250);
      delay(400); // realiza el movimiento durante 1 segundo
      Serial.println("parada");
       digitalWrite (dirA, LOW);
       digitalWrite (dirB, LOW);
      delay(300); // realiza el movimiento durante 300 milisegundos
      k++;
      break;

    }

  } while (vector[k]!=0);

Serial.println("FINISH"); //para los motores
  digitalWrite (dirA, LOW);
  digitalWrite (dirB, LOW);
  analogWrite (speedA, 0);
  analogWrite (speedB, 0);
```

```
      lcd.setCursor(7, 1);
      lcd.print("FINISH");
        delay(100); // realiza el movimiento durante 1 segundo
      digitalWrite(LED2, HIGH);  // Encendemos el LED1 para acabar el movimiento
      delay(1000);
      digitalWrite(LED2, LOW);
            lcd.setCursor(0, 1);
        lcd.print("                ");
      break;

      case 7:
        Serial.println("BORRADO");
            lcd.setCursor(7, 1);
        lcd.print("CLEAR ");
         for (int j=0; j<9; j++) { vector[j]=0; }  //Llenamos de ceros el vector
          digitalWrite(LED1, HIGH);  // Encendemos los LEDs e inicar el borrado
          digitalWrite(LED2, HIGH);
          delay(500);
          digitalWrite(LED1, LOW);
          digitalWrite(LED2, LOW);
        i=0;
        lcd.setCursor(6, 1);
        lcd.print("      ");
        lcd.setCursor(0, 2);
        lcd.print("                ");
        lcd.setCursor(0,3);
        lcd.print("    ARRAY CLEAR     ");
        delay(1000);
        lcd.setCursor(0, 3);
        lcd.print("                ");
      break;

      case 9:
       lcd.setCursor(0, 2);
       lcd.print("Array:");
        lcd.setCursor(6, 2);
        for (int h=0;h<9;h++){
              if (vector[h]==2) {      lcd.print("U,"); }
              if (vector[h]==4) {      lcd.print("L,"); }
              if (vector[h]==6) {      lcd.print("R,"); }
              if (vector[h]==8) {      lcd.print("D,"); }
           } // Saca vector en pantalla
      break;
    }
  }
}
```

D. “Datasheet” de los componentes.

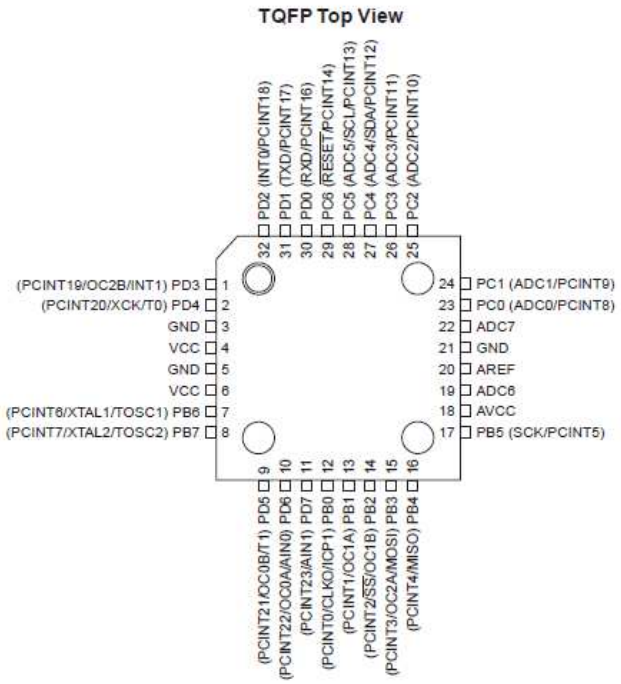
- **8-bit Microcontroller with 4/8/16/32K Bytes In- System Programmable Flash ATmega48PA, ATmega88PA, ATmega168PA, ATmega328P**

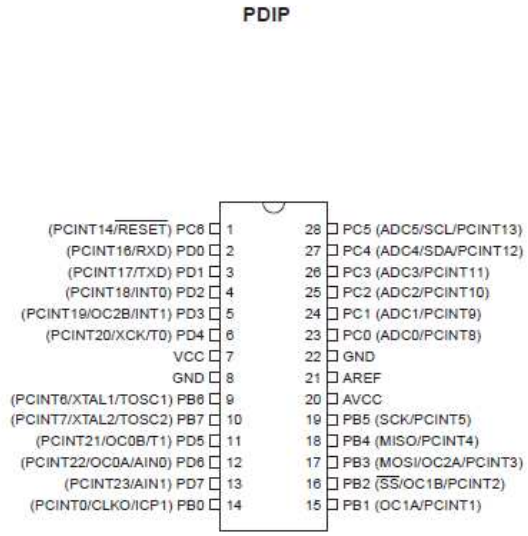
Features

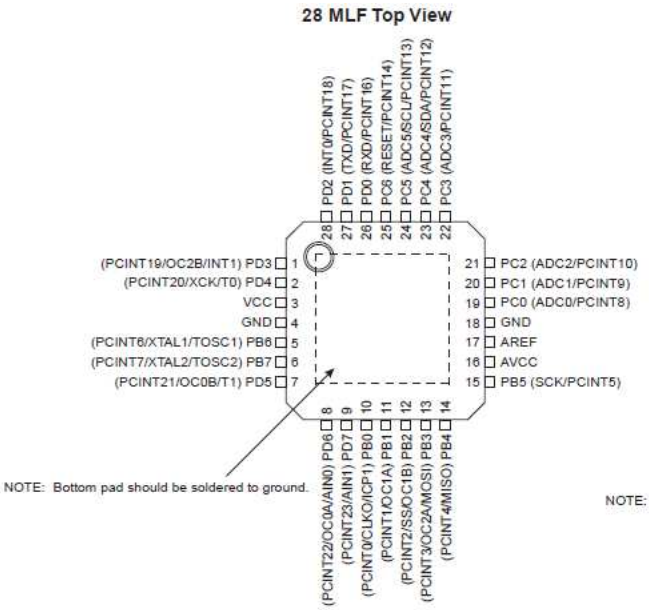
• High Performance, Low Power AVR® 8-Bit Microcontroller
• Advanced RISC Architecture
  – 131 Powerful Instructions – Most Single Clock Cycle Execution
  – 32 x 8 General Purpose Working Registers
  – Fully Static Operation
  – Up to 20 MIPS Throughput at 20 MHz
  – On-chip 2-cycle Multiplier
• High Endurance Non-volatile Memory Segments
  – 4/8/16/32K Bytes of In-System Self-Programmable Flash progam memory (ATmega48PA/88PA/168PA/328P)
  – 256/512/512/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
  – 512/1K/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
  – Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  – Data retention: 20 years at 85°C/100 years at 25°C(1)
  – Optional Boot Code Section with Independent Lock Bits In-System Programming by On-chip Boot Program True Read-While-Write Operation
  – Programming Lock for Software Security
• Peripheral Features
  – Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  – One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  – Real Time Counter with Separate Oscillator
  – Six PWM Channels
  – 8-channel 10-bit ADC in TQFP and QFN/MLF package Temperature Measurement
  – 6-channel 10-bit ADC in PDIP Package Temperature Measurement
  – Programmable Serial USART
  – Master/Slave SPI Serial Interface
  – Byte-oriented 2-wire Serial Interface (Philips I2C compatible)
  – Programmable Watchdog Timer with Separate On-chip Oscillator
  – On-chip Analog Comparator
  – Interrupt and Wake-up on Pin Change
• Special Microcontroller Features
  – Power-on Reset and Programmable Brown-out Detection
  – Internal Calibrated Oscillator
  – External and Internal Interrupt Sources
  – Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
• I/O and Packages
  – 23 Programmable I/O Lines
  – 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
• Operating Voltage:
  – 1.8 - 5.5V for ATmega48PA/88PA/168PA/328P
• Temperature Range:
  – -40°C to 85°C
• Speed Grade:
  – 0 - 20 MHz @ 1.8 - 5.5V
• Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:
  – Active Mode: 0.2 mA
  – Power-down Mode: 0.1 µA
  – Power-save Mode: 0.75 µA (Including 32 kHz RTC)

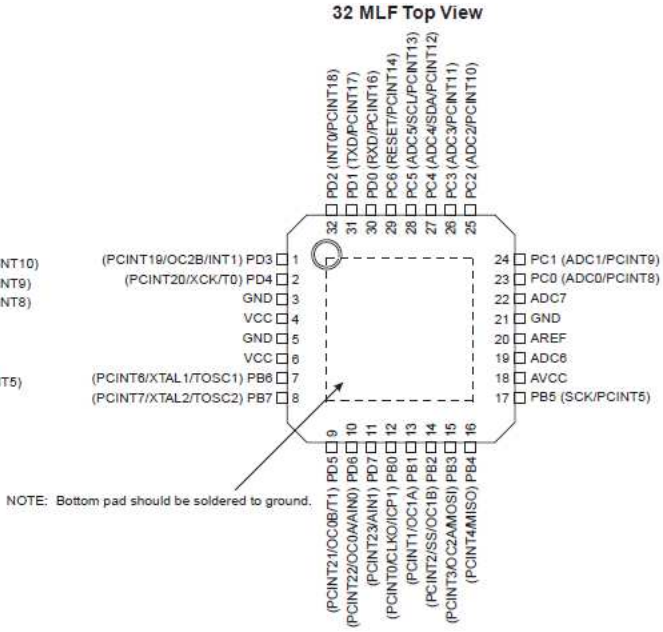
1. Pin Configurations

Figura1. Pinout ATmega48PA/88PA/168PA/328P









2 Pin Descriptions

2.1 VCC

Digital supply voltage.

2.2 GND

Ground.

2.3 Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

The various special features of Port B are elaborated in ”Alternate Functions of Port B” on page 82 and ”System Clock and Clock Options” on page 26.

2.4 Port C (PC5:0)

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5..0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

2.5 PC6/RESET

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running.

The minimum pulse length is given in Table 28-3 on page 318. Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in ”Alternate Functions of Port C” on page 85.

2.6 Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

The various special features of Port D are elaborated in ”Alternate Functions of Port D” on page 88.

2.7 AVCC

AVCC is the supply voltage pin for the A/D Converter, PC3:0, and ADC7:6. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter. Note that PC6..4 use digital supply voltage, VCC.

2.8 AREF

AREF is the analog reference pin for the A/D Converter.

2.9 ADC7:6 (TQFP and QFN/MLF Package Only)

In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter.

These pins are powered from the analog supply and serve as 10-bit ADC channels.

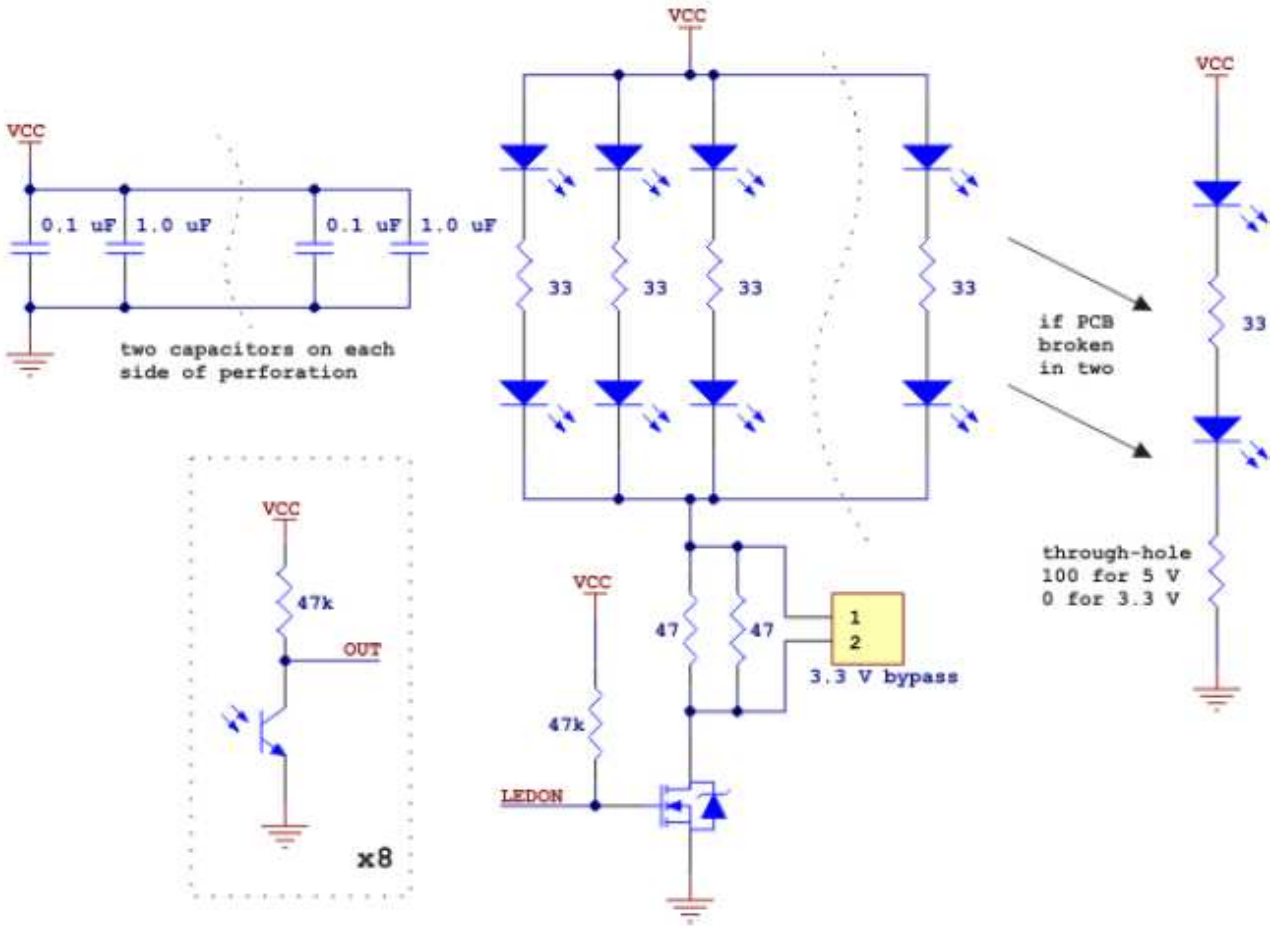
- **QTR-8A Reflectance Sensor Array**

1. Functional Description

The QTR-8A reflectance sensor array is intended as a line sensor, but it can be used as a general-purpose proximity or reflectance sensor. The module is a convenient carrier for eight IR emitter and receiver (phototransistor) pairs evenly spaced at intervals of 0.375" (9.525 mm). Each phototransistor is connected to a pull-up resistor to form a voltage divider that produces an analog voltage output between 0 V and VIN (which is typically 5 V) as a function of the reflected IR. Lower output voltage is an indication of greater reflection.

The outputs are all independent, but the LEDs are arranged in pairs to halve current consumption. The LEDs are controlled by a MOSFET with a gate normally pulled high, allowing the LEDs to be turned off by setting the MOSFET gate to a low voltage. Turning the LEDs off might be advantageous for limiting power consumption when the sensors are not in use or for varying the effective brightness of the LEDs through PWM control.

The LED current-limiting resistors for 5 V operations are arranged in two stages; this allows a simple bypass of one stage to enable operation at 3.3 V. The LED current is approximately 20-25 mA, making the total board consumption just less than 100 mA. The schematic diagram of the module is shown below:



2. Specifications

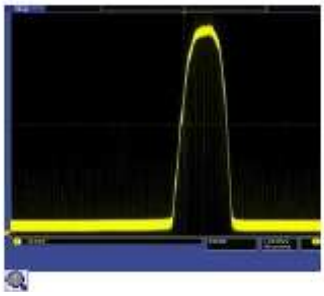
•Dimensions: 2.95" x 0.5" x 0.125" (without header pins installed)
•Operating voltage: 3.3-5.0 V
•Supply current: 100 mA
•Output format: 8 analog voltages
•Output voltage range: 0 V to supplied voltage
•Optimal sensing distance: 0.125" (3 mm)
•Maximum recommended sensing distance: 0.25" (6 mm)
•Weight without header pins: 0.11 oz (3.09 g)

Interfacing with the QTR-8A Outputs

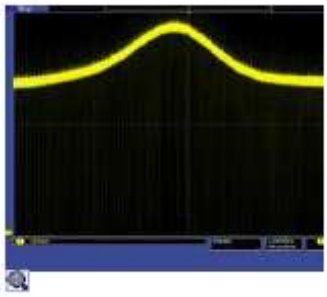
There are several ways you can interface with the QTR-8A outputs:

•Use a microcontroller's analog-to-digital converter (ADC) to measure the voltages.
•Use a comparator with an adjustable threshold to convert each analog voltage into a digital (i.e. black/white) signal that can be read by the digital I/O line of a microcontroller.
•Connect each output directly to a digital I/O line of a microcontroller and rely upon its internal comparator.

This last method will work if you are able to get high reflectance from your white surface as depicted in the he left image, but will probably fail if you u have a lower reflectance signal profile like the one on the right.



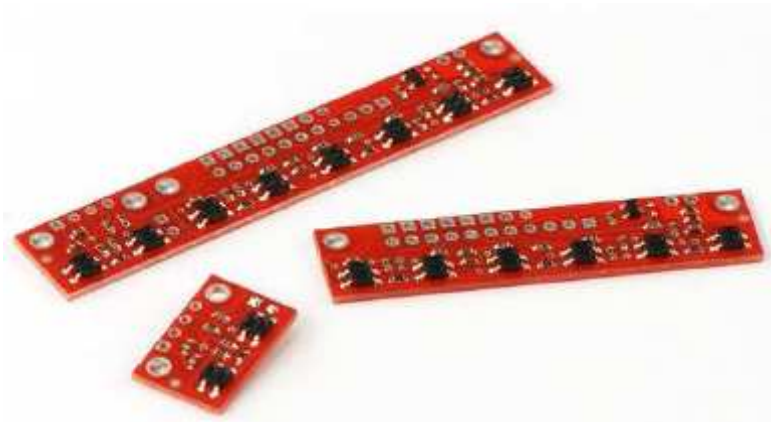
QTR-1A output 1/8" away from a spinning white disk with a black line on it.



QTR-1A output 3/8" away from a spinning white disk with a black line on it.

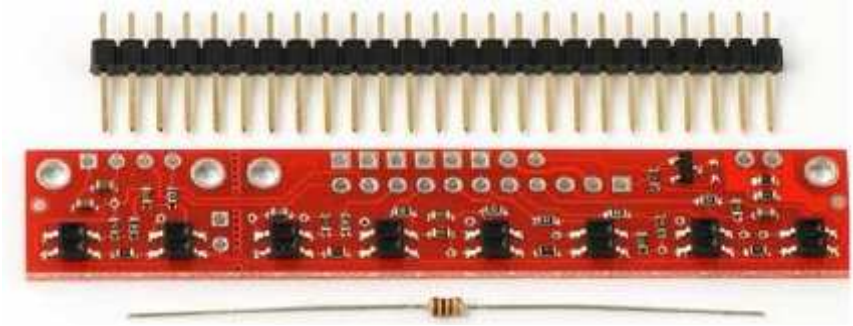
Breaking the Module in Two

If you don't need or can not fit all eight sensors, you can break off two sensors and still use all 8 sensors as two separate modules, as shown below. The PCBB can be scored from both sides along the p perforation and then bent until it snaps apart. . Each of the two resulting pieces will function as an independent line sensor.



Included Components

This module ships with a 25-pin 0.1" header strip and a 100 Ohm through-hole resistor as shown below.



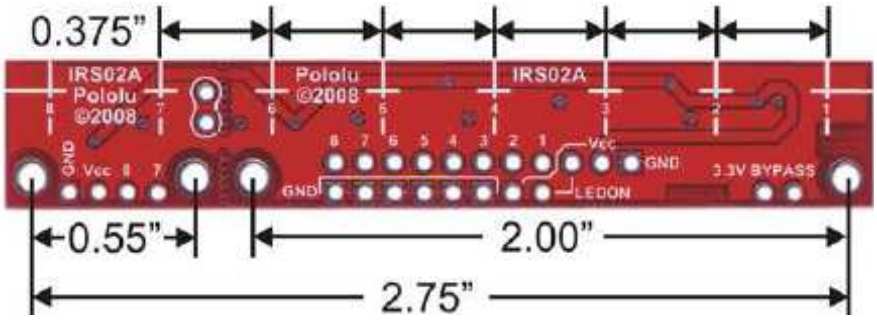
You can break the header strip into smaller pieces and solder them onto your reflectance sensor array as desired, or you can solder wires directly to the unit or use a right-angle header strip for a more compact installation. The pins on the module are arranged so that they can all be accessed using either an 11×1 strip or an 8×2 strip.

The resistor is required to make the two-sensor array functional after the original eight sensor array is broken into two pieces. This resistor is only needed once the board has been broken.

3. Physical Characteristics

Both the QTR-8A and QTR-8RC have the same physical characteristics. Each array measures 2.95" long by 0.50" wide and has a maximum thickness of approximately 0.125". Without header pins, the unit weighs 0.11 oz (3.1 g).

The individual reflectance sensors are distributed evenly on the board at 0.375" intervals. The mounting holes are intended for #2 screws.



QTR-8A and QTR-8RC reflectance sensor array dimensions.

4. Distinguishing the QTR-8A from the QTR-8RC

The QTR-8A sensor array has a different output format from the QTR-8RC, so it is important you know which sensor model you have. What distinguishes one model from another are the components placed on a common PCB.

The pictures below show the two models:

QTR-8A reflectance sensor array.

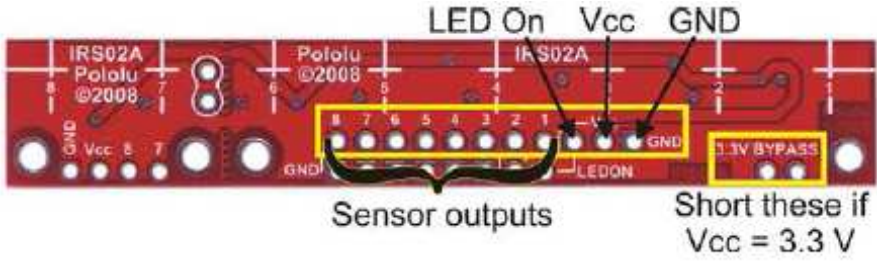
QTR-8RC reflectance sensor array.

The QTR-8A uses a voltage divider for each sensor's output while the QTR-8RC uses an RC circuit. This produces the most visible difference between the two: eight 47k voltage divider resistors (black components with "473" written on them in yellow) on the QTR-8A are replaced by capacitors (tan components) on the QTR-8RC.

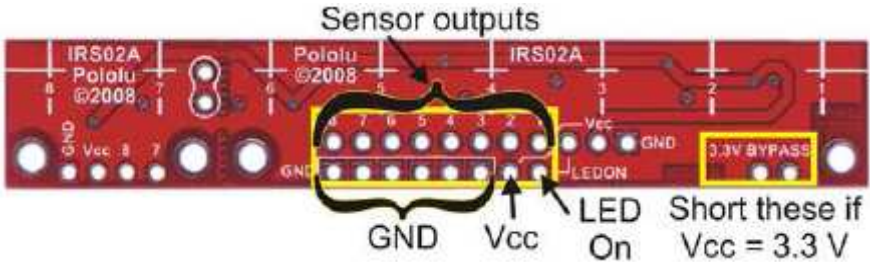
5. Module Connections

The QTR-8x reflectance sensor arrays are designed to provide some connection flexibility. The pins are standard

0.1" spacing and are arranged to support connection using either an 11×1 header strip or an 8×2 header strip. A 25-pin 0.1" header strip is included with the module; you can break this strip into smaller strips and solder them as desired, or you can solder wires directly to the unit for a more compact installation. The two connection methods are pictured below:



QTR-8x reflectance sensor array with 11×1 connection pins labelled.



QTR-8x reflectance sensor array with 8×2 connection pins labelled.

Several pins appear in multiple locations, but you can leave duplicate connection points disconnected. For example, you do not need to connect ground to all six ground pins along the lower edge of the board. Duplicate pins exist so that you can choose the most convenient points to make your connections.

Vcc, GND, and 3.3V BYPASS

The Vcc and GND pins are where the sensor array receives its power. Your Vcc connection must be between 3.3 and 5 V and must be able to supply at least 100 mA. The sensor's outputs will be relative to GND.

The module is calibrated so that its IR LEDs will achieve optimal brightness when Vcc is 5 V; using a lower Vcc will decrease the LED brightness or cause them to turn off completely. You can compensate for this by shorting the two 3.3V BYPASS pins together, which bypasses one stage of the LED current-limiting resistors and increases LED brightness.

Note: Do not short the 3.3V BYPASS pins together while using a Vcc of 5 V. While this would make the LEDs even brighter, it would exceed their current rating and put them at risk of burning out.

LEDON

This pin is connected to a MOSFET that delivers power to the IR LEDs, so its state determines the whether the LEDs are on or off. When this pin is driven high or left disconnected (it is internally pulled high), the LEDs are all on; when this pin is driven low, the LEDs are off. You can use this pin to save power by turning the LEDs off while you are not taking a reading. Additionally, you can connect this pin to a high-frequency PWM to control effective LED brightness and decrease power consumption.

QTR-8A Sensor Outputs

The QTR-8A reflectance sensor array has eight distinct sensor outputs, one from each LED/phototransistor pair.

These outputs are analog voltages ranging from 0 V to Vcc (which must be between 3.3 and 5 V). With a strong reflectance, such as when the sensor is over a white surface, its output voltage will tend towards 0 V; with very weak reflectance, such as when the sensor is over a black surface, its output voltage will tend towards Vcc. To get a good range of readings between white and black surfaces, we recommend you mount your sensor no more than 0.25" away from the surface. In our tests, the optimal distance between the sensor and the surface has been 0.125". Please see our QTR Sample Output Data document [http://www.pololu.com/docs/0J13] for more information.

QTR-8RC Sensor Outputs

The QTR-8RC reflectance sensor array also has eight distinct sensor outputs, one from each LED/phototransistor pair. In the 8RC sensor model, each phototransistor uses a capacitor discharge circuit that allows a digital I/O line on a microcontroller to take an analog reflectance reading by measuring the discharge time of the capacitor. This format has several advantages over the 8A sensor model:

• No analog-to-digital converter (ADC) is required
• Improved sensitivity over voltage-divider analog output
• Parallel reading of all eight sensors is possible with most microcontrollers
• Parallel reading allows optimized use of LED power enable option

When you have a microcontroller digital I/O connected to a sensor output, the typical sequence for reading that sensor is:

1. Turn on IR LEDs (optional)
2. Set the I/O line to an output and drive it high
3. Allow at least 10 us for the 10 nF capacitor to charge
4. Make the I/O line an input (high impedance)
5. Measure the time for the capacitor to discharge by waiting for the I/O line to go low
6. Turn off IR LEDs (optional)

These steps can typically be executed in parallel on multiple I/O lines. With a strong reflectance, such as when a sensor is over a white surface, the discharge time can be as low as several dozen microseconds; with very weak reflectance, such as when the sensor is over a black surface, the discharge time will typically be a few milliseconds, but can be several dozen milliseconds in the worst case. The exact time of the discharge depends on your microcontroller's I/O line characteristics. Meaningful results can be available within 1 ms in typical cases (i.e. when not trying to measure subtle differences in low-reflectance scenarios), allowing up to 1 kHz sampling of all 8 sensors. If lower-frequency sampling is sufficient, substantial power savings can be realized by using the LEDON pin to turn off the LEDs. For example, if a 100 Hz sampling rate is acceptable, the LEDs can be off 90% of the time, lowering average current consumption from 100 mA to 10 mA.

If you want to achieve high frequency sampling rates, we recommend you mount your sensors 0.125" away from the surface. The greater your distance from the surface, the lower the overall reflectance will be, which in turn will result in longer capacitor discharge times. We recommend you mount your sensors no more than 0.375" away from the surface. Please see our QTR Sample Output Data document [http://www.pololu.com/docs/0J13] for more information.

Additional Considerations

As the sensing distance of your sensors increases, the distinction between black and white surfaces starts to decrease. This problem can be exacerbated by infrared components of ambient lighting. The QTR-8x PCB was designed to help

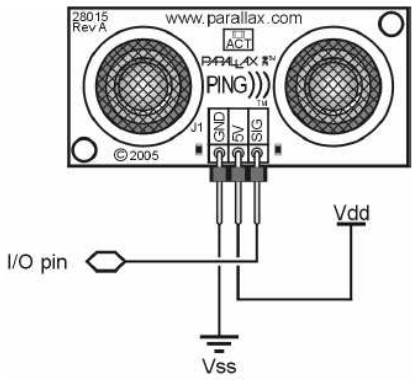
shield the sensors from ambient IR, but this may not be sufficient for areas with unusually high ambient IR or in cases where you need to mount your QTR-8x higher than the recommended 0.25" maximum. In these situations, you might fight it helpful to use shielding (e.g. a skirt around your robot) to block ambient IR.

When troubleshooting your QTR-8x, a digital camera can be useful in determining whether the IR LEDs are functioning as expected. Many older/cheaper digital cameras lack IR filters (most cell phone cameras fit in this category) and can therefore be used to determine if the LEDs are on or off. However, some cameras (usually more expensive ones) have better filters that block IR. In the picture below, a point-and-shoot camera shows the IR LEDs of our IR Beacon [http://www.pololu.com/catalog/product/701] shining (in purple) but the DSLR with which the picture is taken does not show the LEDs.

- **PING)))™ Ultrasonic Distance Sensor (#28015)**

The Parallax PING))) ultrasonic distance sensor provides precise, non-contact distance measurements from about 2 cm (0.8 inches) to 3 meters (3.3 yards). It is very easy to connect to BASIC Stamp® or Javelin Stamp microcontrollers, requiring only one I/O pin.
The PING))) sensor works by transmitting an ultrasonic (well above human hearing range) burst and providing an output pulse that corresponds to the time required for the burst echo to return to the sensor. By measuring the echo pulse width the distance to target can easily be calculated.

Features
• Supply Voltage – 5 VDC
• Supply Current – 30 mA typ; 35 mA max
• Range – 2 cm to 3 m (0.8 in to 3.3 yrds)
• Input Trigger – positive TTL pulse, 2 uS min, 5 µs typ.
• Echo Pulse – positive TTL pulse, 115 uS to 18.5 ms
• Echo Hold-off – 750 µs from fall of Trigger pulse
• Burst Frequency – 40 kHz for 200 µs
• Burst Indicator LED shows sensor activity
• Delay before next measurement – 200 µs
• Size – 22 mm H x 46 mm W x 16 mm D (0.84 in x 1.8 in x 0.6 in)

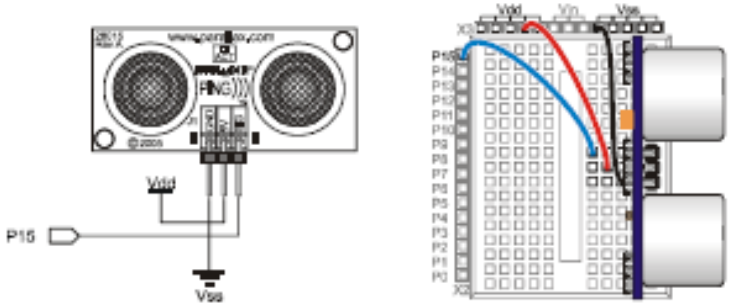


Pin Definitions

GND - Ground (Vss)
5 V - 5 VDC (Vdd)
SIG - Signal (I/O pin)

The PING))) sensor has a male 3-pin header used to supply power (5 VDC), ground, and signal. The header allows the sensor to be plugged into a solderless breadboard, or to be located remotely through the use of a standard servo extender cable (Parallax part #805-00002). Standard connections are show in the diagram to the right.

Quick-Start Circuit
This circuit allows you to quickly connect your PING))) sensor to a BASIC Stamp® 2 via the Board of Education® breadboard area. The PING))) module's GND pin connects to Vss, the 5 V pin connects to Vdd, and the SIG pin connects to I/O pin P15. This circuit will work with the example program Ping_Demo.BS2 listed on page 7.

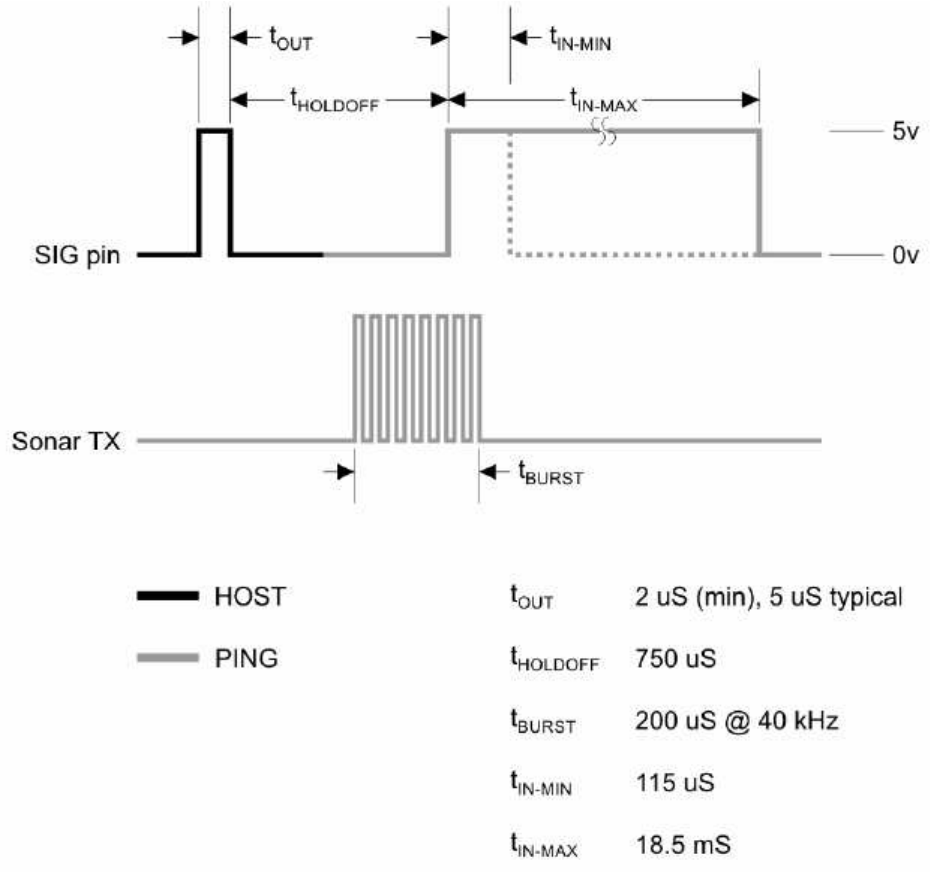


Servo Cable and Port Cautions

If you want to connect your PING))) sensor to a Board of Education using a servo extension cable, follow these steps:
1. When plugging the cable onto the PING))) sensor, connects Black to GND, Red to 5 V, and White to SIG.
2. Check to see if your Board of Education servo ports have a jumper, as shown at right.
3. If your Board of Education servo ports have a jumper, set it to Vdd as shown.
4. If your Board of Education servo ports do not have a jumper, do not use them with the PING))) sensor. These ports only provide Vin, not Vdd, and this may damage your PING))) sensor. Go to the next step.
5. Connect the servo cable directly to the breadboard with a 3-pin header. Then, use jumper wires to connect Black to Vss, Red to Vdd, and White to I/O pin P15.

Theory of Operation

The PING))) sensor detects objects by emitting a short ultrasonic burst and then "listening" for the echo. Under control of a host microcontroller (trigger pulse), the sensor emits a short 40 kHz (ultrasonic) burst. This burst travels through the air at about 1130 feet per second, hits an object and then bounces back to the sensor. The PING))) sensor provides an output pulse to the host that will terminate when the echo is detected, hence the width of this pulse corresponds to the distance to the target.

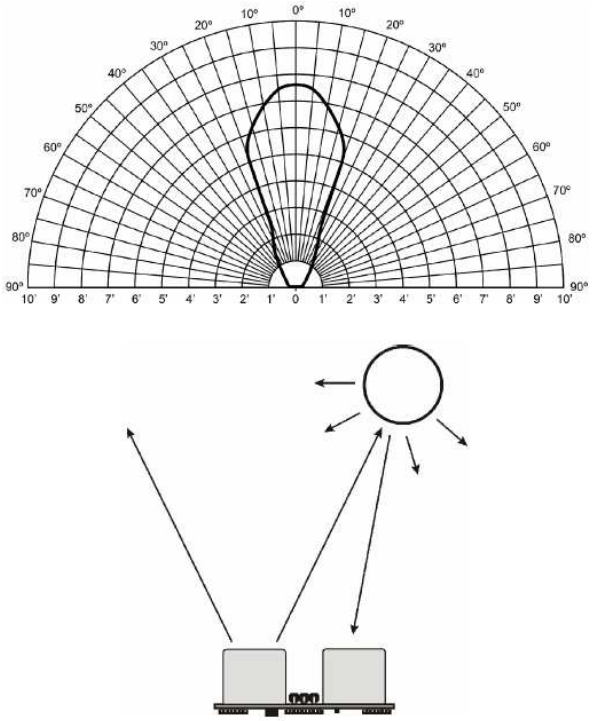


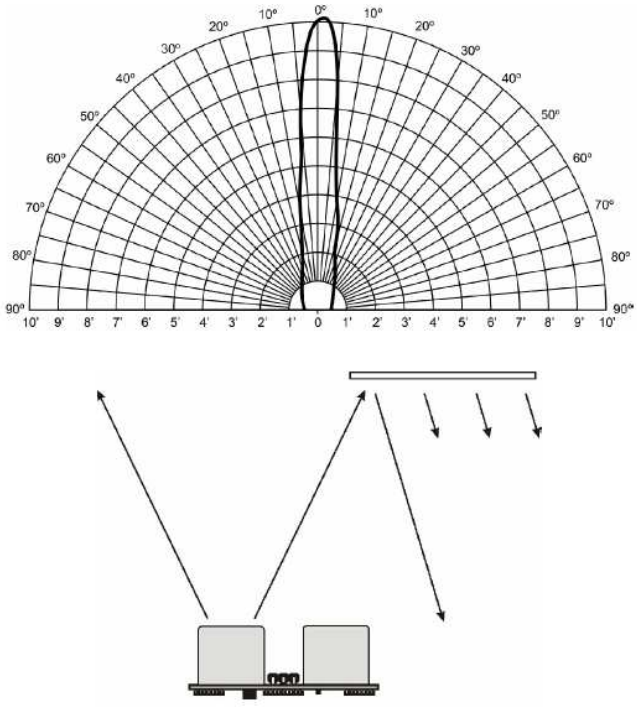
Test Data

The test data on the following pages is based on the PING))) sensor, tested in the Parallax lab, while connected to a BASIC Stamp microcontroller module. The test surface was a linoleum floor, so the sensor was elevated to minimize floor reflections in the data. All tests were conducted at room temperature, indoors, in a protected environment. The target was always centred at the same elevation as the PING))) sensor.

Test 1
Sensor Elevation: 40 in. (101.6 cm)
Target: 3.5 in. (8.9 cm) diameter cylinder, 4 ft. (121.9 cm) tall – vertical orientation

Test 2
Sensor Elevation: 40 in. (101.6 cm)
Target: 12 in. x 12 in. (30.5 cm x 30.5 cm) cardboard, mounted on 1 in. (2.5 cm) pole
- target positioned parallel to backplane of sensor





Test 1 Test 2

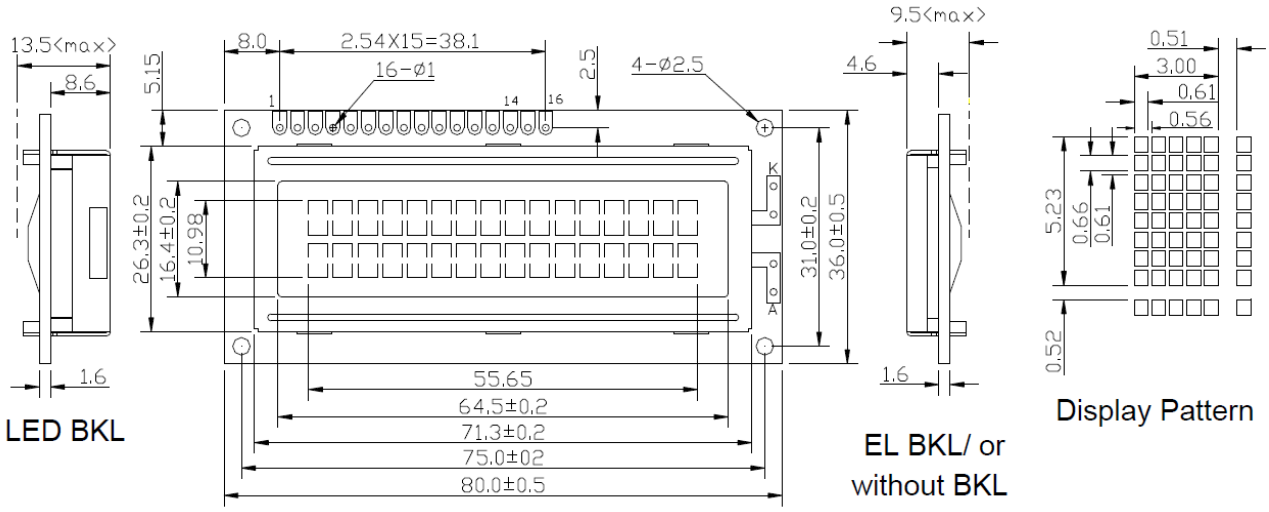
- **GWS MINI L Mini Servo**

The GWS MINI L servo delivers the performance of a standard servo in a smaller and lighter package. The MINI L is slightly smaller, faster, and more powerful than the MINI.

Dimensions

Size: 35.0 x 16.9 x 27.0 mm
Weight: 22 g
General specifications
Digital: No
Speed @ 6V: 0.11 sec/60°
Torque @ 6V: 4.4 kg·cm
Speed @ 4.8V: 0.14 sec/60°
Torque @ 4.8V: 3.6 kg·cm
Lead length: 130 mm
Hardware included: Yes

- **XIAMEN OCULAR GDM1602K**



Feature

1. *5X8 dots with cursor*
2. *Built-in controller (KS0066U or Equivalent)*
3. *+5V power supply(Also available for +3.0V)*
4. *1/16 duty*
5. *BKL to be driven by pin1,pin2,or pin15,pin16,or A,K*
6. *N.V.optional*

PIN NO	Symbol	Fuction
1	VSS	GND
2	VDD	+5V
3	V0	Contrast adjustment
4	RS	H/L Register select signal
5	R/W	H/L Read/Write signal
6	E	H/L Enable signal
7	DB0	H/L Data bus line
8	DB1	H/L Data bus line
9	DB2	H/L Data bus line
10	DB3	H/L Data bus line
11	DB4	H/L Data bus line
12	DB5	H/L Data bus line
13	DB6	H/L Data bus line
14	DB7	H/L Data bus line
15	A	+4.2V for LED
16	K	Power supply for BKL(0V)

Mechanical Data

Item	Standard	Unit
Module dimension	*80.0x36.0*	*mm*
Viewing area	*64.5x16.4*	*mm*
Dot size	*0.56x0.61*	*mm*
Character size	*3.00x5.23*	*mm*

Absolute Maximum Rating

Item	Symbol	Standard			Unit
		Min	Typ	Max	
Power supply	*VDD-VSS*	-0.3	--------	5.5	V
Input voltage	*VI*	-0.3	--------	VDD	

Electronical characteristics

Item	Symbol	Condition	Standard			Unit
			Min	Typ	Max	
Input voltage	*VDD*	+5V	4.7	5.0	5.5	V
		+3.3V	2.7	3.0	5.3	V
Supply current	*IDD*	VDD=5V	------	1.5	4	mA
Recommended LCD riling voltage for normal temp version module	VDD-V0	-20ºC	------	------	------	V
		0 ºC	4.7	5.0	5.5	
		25ºC	4.3	4.5	4.7	
		50ºC	4.1	4.3	4.5	
		70ºC	------	-------	------	
LED forward voltage	*VF*	25ºC	------	4.2	4.6	V
LED forward current	*IF*	25ºC	------	120	160	mA
EL power supply current	IEL	VEL=110V AC 400Hz	-----	-------	------	mA

Display character address code:

Display position

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
DDRAM address	*00*	*01*	*02*	---	---	---	---	---	---	---							*0FH*
DDRAM address	*40*	*41*	*42*	---	---	---	---	---	---	---							*4FH*

INSTRUCCIONES DE INSTALACION ARDUINO

Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos.
Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y puede afectar aquello que le rodea controlando luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software (p.ej. Flash, Processing, MaxMSP).
Las placas pueden ser hechas a mano o compradas montadas de fábrica; el software puede ser descargado de forma gratuita. Los ficheros de diseño de referencia (CAD) están disponibles bajo una licencia abierta, así pues eres libre de adaptarlos a tus necesidades.
Arduino recibió una Mención Honorífica en la sección Digital Communities de la edición del 2006 del Ars Electronica Prix. El equipo Arduino (Arduino team) es: Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, y David Mellis.

- Compra una placa Arduino
- Descarga el Software Arduino

Desarrollo: Para información en el desarrollo de Arduino, visita el proyecto Arduino en Google Code. Los cambios a realizar en el software se discuten en la lista de correo de desarrolladores (en inglés).
En cualquier sitio: podrás encontrar fotografías de proyectos y talleres hechos con Arduino, por ejemplo siguiendo el marcador de Arduino en Flicker. Podrás encontrar enlaces relacionados con el tema en el marcador Arduino.

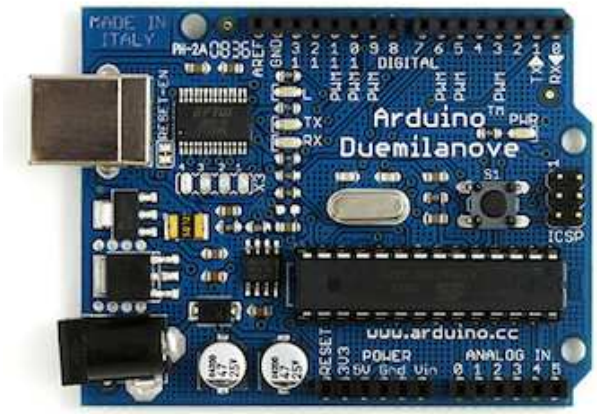
Primeros Pasos
Para comenzar, sigue las instrucciones que correspondan a tu sistema operativo: Windows, Mac OS X o Linux; o las referentes a tu placa: Arduino Nano, Arduino Mini, Arduino BT, LilyPad Arduino, XBee shield. Si encontrases problemas, echa un vistazo a resolución de problemas para sugerencias.

Comenzando con Arduino en Windows

Este documento explica cómo conectar tu placa Arduino al ordenador y volcar el código de tu primer programa.

1 | Consigue un Arduino y un cable USB
2 | Descarga el IDE de Arduino
3 | Conecta la placa
4 | Instala los *drivers*
5 | Ejecuta la Aplicación Arduino
6 | Abre el ejemplo *Blink*
7 | Selecciona tu placa
8 | Selecciona tu puerto serie
9 | Sube el *sketch* a la placa

1 | Consigue un Arduino y un cable USB
En este tutorial asumimos que estás usando una placa Arduino Duemilanove, Nano o Diecimila. Si tienes cualquier otra placa necesitas leer la página correspondiente a la placa que uses en esta guía de iniciación.
También necesitarás un cable estándar USB (conexión A a conexión B), como los que se usan para conectar, por ejemplo, una impresora USB. (En el caso de la placa Arduino Nano necesitarás un cable de conexión A a conexión mini-B).



Fotos cortesía de SparkFun Electronics.

2 | Descarga el IDE de Arduino

Descarga la última versión de la página de descargas.
Cuando la descarga finalice, descomprime el fichero. Asegúrate de mantener la estructura de directorios. Haz doble click en la carpeta *arduino-00XX* para abrirla. Deberías ver una serie de ficheros y carpetas ahí dentro.

3 | Conecta la placa

Conecta la placa Arduino a tu ordenador usando el cable USB. el LED verde indicador de la alimentación (nombrado como **PWR** en la placa) debería quedar encendido a partir de ese momento.
Si estás usando una placa Arduino Diecimila, necesitarás asegurarte de que la placa está configurada para alimentarse mediante la conexión USB. La fuente de alimentación se selecciona con un puente ("jumper"), una pequeña pieza de plástico que encaja en dos de los tres pins situados entre los conectores USB y de alimentación de la placa. Comprueba que el puente esté conectando los dos pins más cercanos al puerto USB de la placa.
En las placas Arduino Duemilanove y Arduino Nano la fuente de alimentación adecuada se selecciona de forma automática y no requiere de realizar ninguna comprobación en este sentido.

4 | Instala los *drivers*

Cuando conectas la placa, Windows debería inicializar la instalación de los *drivers* (siempre y cuando no hayas utilizado ese ordenador con una placa Arduino anteriormente).
En Windows Vista y Windows 7, los drivers deberían descargarse e instalarse automáticamente.
En Windows XP, se abrirá el diálogo de instalación de Nuevo Hardware:

- Cuando te pregunten: ¿**Puede Windows conectarse a Windows Update para buscar el software**? selecciona "**No, no esta vez**". Haz click en siguiente.
- Selecciona "**Instalar desde una lista o localización específica (Avanzado)**" haz click en siguiente.
- Asegúrate que "**Buscar los mejores drivers en estas localizaciones**" está seleccionado; deselecciona "**Buscar en medios removibles**"; selecciona "**Incluye esta localización en la búsqueda**" y navega al directorio **drivers/FTDI USB Drivers** dentro de la carpeta de Arduino que has descomprimido previamente. (La versión más reciente de los drivers se puede encontrar en la página web del fabricante del chip FTDI). Haz click en siguiente.
- El asistente de instalación buscará los drivers y te anunciará que encontró un "USB Serial Converter" (se traduce por Conversor USB-Serie). Haz click en finalizar.
- El asistente de instalación de hardware volverá a iniciarse. Repite los mismos pasos que antes y selecciona la misma carpeta de instalación de los drivers. Esta vez encontrará un "USB Serial Port" (o puerto USB-Serie).

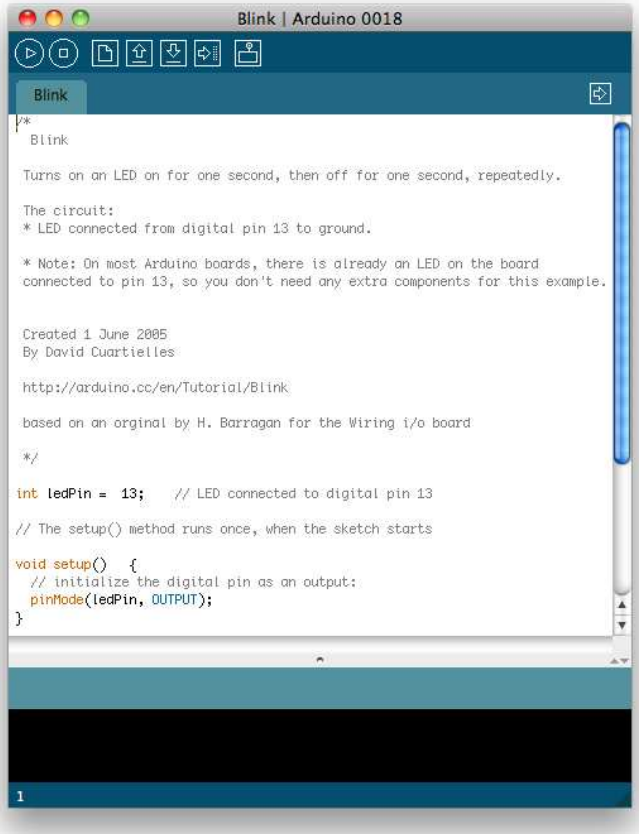
Puedes comprobar que los *drivers* se han instalado correctamente abriendo la carpeta del Administrador del Dispositivos, en el grupo *Dispositivos* del panel de control del sistema. Busca "USB Serial Port" (o *Puerto USB-Serie*) en la sección de puertos; esa es tu placa Arduino.

5 | Ejecuta la Aplicación Arduino

Haz doble click en la aplicación Arduino.

6 | Abre el ejemplo *Blink*

Abre el programa de ejemplo para hacer parpadear un LED ("LED blink"): **File > Examples > Digital > Blink**.



7 | Selecciona tu placa

Necesitarás seleccionar el tipo de placa de tu Arduino en el menú **Tools > Board**. Para las nuevas placas Arduino con el chip ATmega 328 (comprueba el texto escrito en el chip de la placa), selecciona la opción **Arduino Duemilanove o Nano w/ ATmega328** del menú desplegable. Anteriormente las placas Arduino incluían un chip ATmega 168; para estos casos selecciona la opción **Arduino Diecimila, Duemilanove, or Nano w/ ATmega168**. (Se puede encontrar más detalles sobre los dispositivos de entrada de las placas en el menú desplegable en la página del entorno arduino.)

8 | Selecciona tu puerto serie

Selecciona el dispositivo serie de la placa Arduino en el menú Tools | Serial Port (*Herramientas* | *Puertos Serie*). Lo más probable es que sea **COM3** o mayor (**COM1** y **COM2** se reservan, por regla general para puertos serie de hardware). Para asegurarte de cual es, puedes desconectar la placa y volver a mirar el menú; el puerto de la placa habrá desaparecido de la lista. Reconecta la placa y selecciona el puerto apropiado.

9 | Sube el *sketch* a la placa

Ahora simplemente pulsa sobre el botón "Upload" en el Entorno Arduino. Espera unos pocos segundos - deberías ver parpadear los led RX y TX de la placa. Si el volcado del código es exitoso verás aparecer el mensaje "Done uploading" en la barra de estado. (*Aviso:* Si tienes una placa Arduino Mini, NG, u otras placas, necesitarás presionar el botón de reseteo de la placa inmediatamente antes de presionar el botón "Upload" el Entorno de programación Arduino.)

Unos pocos segundos después de finalizar el volcado del programa deberías ver cómo el led de la placa conectado al pin 13 (L) comienza a parpadear (con un color naranja). Si ocurre esto ¡enhorabuena! Ya tienes tu Arduino listo y funcionando.

Si tienes problemas, por favor, consulta nuestras sugerencias ante problemas. A partir de ahora también podrás encontrar interesante consultar:

- los ejemplos sobre el funcionamiento de distintos sensores y dispositivos.
- la sección referencia para conocer el lenguaje de programación Arduino.

Todo este texto se encuentra en la página principal de arduino (arduino.cc y arduino.cc/es), con la explicación para todo tipo de sistemas operativo. Nosotros aportamos el de Windows ya que es el más utilizado.

MANUAL LENGUAJE DE PROGRAMACIÓN EN EL ENTORNO ARDUINO

1. ESTRUCTURA

- setup() *(inicialización)*
- loop() *(bucle)*

Estructuras de control

- if *(comparador si-entonces)*
- if...else *(comparador si...sino)*
- for *(bucle con contador)*
- switch case *(comparador múltiple)*
- while *(bucle por comparación booleana)*
- do... while *(bucle por comparación booleana)*
- break *(salida de bloque de código)*
- continue *(continuación en bloque de código)*
- return *(devuelve valor a programa)*

Sintaxis

- ; (punto y coma)
- {} (llaves)
- // (comentarios en una línea)
- /* */ (comentarios en múltiples líneas)

Operadores Aritméticos

- = (asignación)
- + (suma)
- - (resta)
- * (multiplicación)
- / (división)
- % (resto)

Operadores Comparativos

- == (igual a)
- != (distinto de)
- < (menor que)
- > (mayor que)
- <= (menor o igual que)
- >= (mayor o igual que)

Operadores Booleanos

- && (y)
- || (o)
- ! (negación)

Operadores de Composición

- ++ (incrementa)
- -- (decremento)
- += (composición suma)
- -= (composición resta)
- *= (composición multiplicación)
- /= (composición división)

2. VARIABLES

Constantes

- HIGH | LOW
- INPUT | OUTPUT
- true | false

- Constantes Numéricas

Tipos de Datos

- boolean *(booleano)*
- char *(carácter)*
- byte
- int *(entero)*
- unsigned int *(entero sin signo)*
- long *(entero 32b)*
- unsigned long *(entero 32b sin signo)*
- float *(en coma flotante)*
- double *(en coma flotante de 32b)*
- string *(cadena de caracteres)*
- array *(cadena)*
- void *(vacío)*

3. FUNCIONES

E/S Digitales

- pinMode()
- digitalWrite()
- digitalRead()

E/S Analógicas

- analogRead()
- analogWrite() - *PWM (modulación por ancho de pulso)*

E/S Avanzadas

- tone()
- noTone()

- shiftOut()
- pulseIn()

Tiempo

- millis()
- micros()
- delay()
- delayMicroseconds()

Matemáticas

- min() *(mínimo)*
- max() *(máximo)*
- abs() *(valor absoluto)*
- constrain() *(limita)*
- map() *(cambia valor de rango)*
- pow() *(eleva a un número)*
- sq() *(eleva al cuadrado)*
- sqrt() *(raíz cuadrada)*

Trigonometría

- sin() *(seno)*
- cos() *(coseno)*
- tan() *(tangente)*

Números Aleatorios

- randomSeed()
- random()

Comunicación

- Serial

Funciones

- begin()
- end()
- available()
- read()
- flush()
- print()
- println()
- write()

1. ESTRUCTURA:

SETUP()

La función setup() se establece cuando se inicia un programa -sketch. Se emplea para iniciar variables, establecer el estado de los pins, inicializar librerías, etc. Esta función se ejecutará una única vez después de que se conecte la placa Arduino a la fuente de alimentación, o cuando se pulse el botón de reinicio de la placa.

Ejemplo

```
int buttonPin = 3;

void setup()
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}

void loop()
{
  // ...
}
```

LOOP()

Luego de crear la función setup(), la cual inicializa y prepara los valores iniciales, la función loop() hace justamente lo que su nombre sugiere, por lo tanto se ejecuta consecutivamente, permitiéndole al programa variar y responder. Úsala para controlar de forma activa la placa Arduino.

Ejemplo

```
int buttonPin = 3;

// setup inicializa la comunicación serial y el buttonPin
void setup()
{
  beginSerial(9600);
  pinMode(buttonPin, INPUT);
}

// loop obtiene el estado del pin del botón cada vez,
// y de estar presionado, lo comunica por serial.
void loop()
{
  if (digitalRead(buttonPin) == HIGH)
    serialWrite('H');
  else
    serialWrite('L');

  delay(1000);
}
```

ESTRUCTURAS DE CONTROL

IF (CONDICIONAL) Y ==, !=, <, > (OPERADORES DE COMPARACIÓN)

`if`, el cual puede ser usado en conjunto con uno o más operadores de comparación, comprueba si cierta condición se cumple, por ejemplo, si un *input* posee un valor mayor a cierto número. El formato para una comprobación *if* es el siguiente:

```
if (algunaVariable > 50)
{
  // hacer algo aquí.
}
```

Este programa comprueba si la variable *algunaVariable* es mayor a *50*. Si lo es, el programa toma una acción particular. Dicho de otra forma, si la declaración escrita dentro de los paréntesis es verdadera (*true*), el código dentro de las llaves se ejecutará. Sino, el programa ignora dicho código.

Las llaves pueden ser omitidas luego de una declaración *if*. De hacer esto, la siguiente línea (definida por el punto y coma) será la única afectada por la condición.

```
if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)
digitalWrite(LEDpin, HIGH);

if (x > 120){ digitalWrite(LEDpin, HIGH); }

if (x > 120){
  digitalWrite(LEDpin1, HIGH);
  digitalWrite(LEDpin2, HIGH);
}

// todos los ejemplos son correctos.
```

Las declaraciones a evaluar dentro de los paréntesis, requieren el uso de uno o más operadores:

Operadores de Comparación:

```
x == y (x es igual a y)
x != y (x no es igual a y)
x <  y (x es menor a y)
x >  y (x es mayor a y)
x <= y (x es menor o igual a y)
x >= y (x es mayor o igual a y)
```

Atención:

Ten cuidado de no usar un signo de igual solo (ej. `if (x = 10)`). Un signo de igual solo es el operador que indica la asignación de un valor, y va a asignar *10* a *x*. En su lugar usa el signo de igual doble (ej. `if (x == 10)`), el cual es el operador de comparación, y compryeba *si* x equivale a 10 o no. El último ejemplo sólo da *true* si x equivale a 10, pero el ejemplo anterior (con un sólo símbolo =) dará siempre *TRUE*.

Esto es porque C evalúa la declaración `if (x=10)` de la siguiente manera: 10 es asignado a x (Recuerda que un signo = solo, es el operador de asignación), por lo tanto x ahora contiene 10. Entonces el condicional *if* evalúa 10, el cual siempre resulta *TRUE*, debido a que cualquier valor numérico mayor a 0 es evaluado como *TRUE*. Consecuentemente, `if (x = 10)` siempre será evaluado como *TRUE*, lo cual no es el resultado deseado cuando se usa una declaración *if*. Adicionalmente, la variable x será definida en 10, lo que tampoco es una acción deseada.

if también puede ser parte de una estructura de control de ramificación usando la construcción if...else.

IF / ELSE

if/else permite mayor control sobre el flujo del código que la declaración *if* básica, por permitir agrupar múltiples comprobaciones. Por ejemplo, un *input* análogo podría ser comprobado, y tomarse una acción si el valor del *input*es menor a 500, y, otra acción se ser igual o mayor a 500. El código se vería así:

```
if (pinCincoInput < 500)
{
  // acción A
}
else
{
  // acción B
}
```

else puede proceder a una comprobación **if**, de esta forma, se pueden realizar múltiples comprobaciones en una misma estructura de condiciones. Cada comprobación procederá a la siguiente, sólo cuando su propio resultado sea *FALSE*. Cuando el resultado sea *TRUE*, su bloque de código contenido, será ejecutado, y el programa esquivará las siguientes comprobaciones hasta el final de la estructura de comprobaciones. Si ninguna comprobación devuelve valor *TRUE*, el **else** será ejecutado, y de no haber ninguno declarado, simplemente no sucede nada.

Entonces un bloque **else if** puede ser usado con o sin **else** al final. La cantidad de declaraciones **else if**, y sus ramificaciones son ilimitadas.

```
if (pinCincoInput < 500)
{
  // ejecutar A
}
else if (pinCincoInput >= 1000)
{
  // ejecutar B
}
else
{
  // ejecutar C
}
```

Otra forma de expresar ramificaciones (*branching* en inglés), y realizar comprobaciones mutuamente exclusivas, es con la declaración switch case, la cual resulta más idónea en algunos casos.

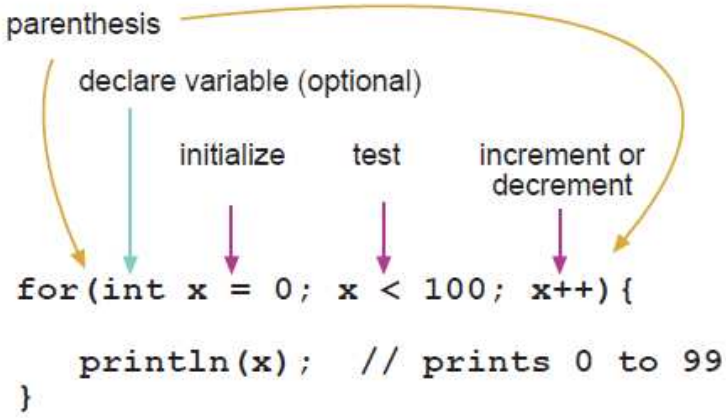
DECLARACION FOR

Descripción

La declaración **for** es usada para repetir un bloque encerrado entre llaves. Un incremento de un contador es usado, normalmente, para aumentar y terminar con el bucle. La estructura **for** es muy útil para la mayoría de las operaciones repetitivas, y habitualmente se usa para operaciones con vectores, para operar sobre conjuntos de datos/pines

El bucle **for** tiene tres partes o argumentos en su inicialización:

```
for (initialization; condition; increment) {
//función(es);
}
```



La **initialization**, o inicialización, se produce sólo la primera vez. Cada vez que se va a repetir el bucle, se revisa la **condition**, o condición: si es cierta, el bloque de funciones (y el incremento del contador) se ejecutan, y la condición vuelve a ser comprobada de nuevo. Si la condición es falsa, el bucle termina.

Ejemplo

```
// Variar la intensidad de un LED usando un salida PWM
int PWMpin = 10; // En el pin 10 hay un LED en serie con una
resistencia de 470 ohmios

void setup()
{
  // no es necesario nada aquí
}

void loop()
{
   for (int i=0; i <= 255; i++){
      analogWrite(PWMpin, i);
      delay(10);
   }
}
```

Consejos de programación

El bucle **for**, en C, es mucho más flexible que otros bucles **for** en otros lenguajes, incluyendo BASIC. Cualquiera (o todos) los parámetros pueden ser omitidos, sin embargo los puntos y coma (;) son obligatorios. También las inicialización, condición e incremento pueden ser cualquier declaración en C válida, con variables independientes, y podemos usar cualquier tipo de variable, incluídos los float. Estos tipos de declaración **for** poco usuales pueden proporcionar una solución válida a algunos problemas de programación raros.

Por ejemplo, usando la multiplicación en el parámetro de incremento, podemos generar una progresión logarítmica.

```
for(int x = 2; x < 100; x = x * 1.5){
   println(x);
}
```

Este código generará: 2,3,4,6,9,13,19,28,42,63,94

Otro ejemplo es hacer apagarse/encenderse un LED poco a poco, Another example, fade an LED up and down with one **for** loop:

```
void loop()
{
   int x = 1;
   for (int i = 0; i > -1; i = i + x){
      analogWrite(PWMpin, i);
      if (i = 255) x = -1;            // cambia de signo para apagarlo
      delay(10);
   }
}
```

SENTENCIA SWITCH / CASE

Como las sentencias **if**, **switch...case** controla el flujo de programas permitiendo a los programadores especificar diferentes códigos que deberían ser ejecutados en función de varias condiciones. En particular, una sentencia switch compara el valor de una variable con el valor especificado en las sentencias case. Cuando se encuentra una sentencia case cuyo valor coincide con dicha variable, el código de esa sentencia se ejecuta.

La palabra clave **break** sale de la sentencia switch, y es usada típicamente al final de cada case. Si una sentencia break, la sentencia switch continuaría ejecuntándo las siguientes expresiones ("falling-through") hasta encontrar un break, o hasta llegar al final de la sentencia switch.

Ejemplo

```
switch (var) {
  case 1:
    //hacer algo cuando sea igual a 1
    break;
  case 2:
    //hacer algo cuando sea igual a 2
    break;
  default:
    // si nada coincide, ejecuta el "default"
    // el "default" es opcional
}
```

Sintaxis

```
switch (var) {
  case etiqueta:
    // sentencias
    break;
  case etiqueta:
    // sentencias
    break;
  default:
    // sentencias
}
```

Parámetros

var: la variable cuyo valor comparas con los varios "case"

etiqueta: un valor para comparar con la variable

BUCLES WHILE

Descripción

Los bucles **while** se ejecutan continuamente, hasta que la expresión de dentro del paréntesis, (), pasa a ser falsa. Algo debe modificar la variable comprobada, el bucle **while** nunca terminará. Lo que modifique la variable puede estar en el código, como una variable que se incrementa, o ser una condición externa, como el valor que da un sensor.

Sintaxis

```
while(expresion){
  // sentencia(s)
}
```

Parámetros

expresion - una sentencia C (booleana) que da como valor verdadero (true) o falso (false)

Ejemplo

```
var = 0;
while(var < 200){
  // haz algo repetitivo 200 veces
  var++;
}
```

DO - WHILE

El bucle "'do'" trabaja de la misma manera que el bucle "'while'", con la excepción de que la condición se comprueba al final del bucle, por lo que este bucle se ejecuta "siempre" al menos una vez.

```
do
{
    // bloque de instrucciones
} while (condición);
```

Ejemplo:

```
do
{
  delay(50);              // espera a que los sensores se estabilicen
  x = readSensors();  // comprueba los sensores

} while (x < 100);    //si se cumple la condición se repite el bucle
```

BREAK

break es usado para salir de los bucles **do**, **for**, o **while**, pasando por alto la condición normal del bucle. Es usado también para salir de una estructura de control **switch**.

Ejemplo

```
for (x = 0; x < 255; x ++)
{
    digitalWrite(PWMpin, x);
    sens = analogRead(sensorPin);
    if (sens > threshold){      // bail out on sensor detect
       x = 0;
       break;   // sale del bucle for.
    }
    delay(50);
}
```

CONTINUE

La sentencia continue omite el resto de iteraciones de un bucle (**do**, **for**, o **while**). Continúa saltando a la condición de bucle y procediendo con la siguiente iteración.

Ejemplo

```
for (x = 0; x < 255; x ++)
{
    if (x > 40 && x < 120){      // crea un salto en estos valores
        continue;
    }

    digitalWrite(PWMpin, x);
    delay(50);
}
```

RETURN

Termina una función y devuelve un valor a la función que la llama. Puede no devolver nada.

Sintaxis

return;

return valor; // ambas formas son correctas

Parámetros

valor: cualquier variable o tipo constante

Ejemplos

Una función que compara la entrada de un sensor a un umbral

```
int comprobarSensor(){
   if (analogRead(0) > 400) {
       return 1;
   else{
       return 0;
   }
}
```

La palabra clave *return* es útil para depurar una sección de código sin tener que comentar una gran cantidad de líneas de código posiblemente incorrecto.

```
void loop(){

// código magnífico a comprobar aquí

return;

// el resto del programa del que se desconfía
// que nunca será ejecutado por estar detrás de return
}
```

SINTAXIS

; PUNTO Y COMA

Utilizado para terminar una declaración.

Ejemplo

```
int a = 13;
```

Truco

Olvidarse de terminar una línea con el punto y coma hará que se lance un error de compilación. El texto del error puede ser obvio y referirse concretamente a la ausencia del punto y coma, pero puede no hacerlo. Si aparece un error de compilación impenetrable y aparentemente incoherente, una de las primeras cosas a hacer es comprobar que no falta ningún punto y coma en la vecindad de la línea en la que el compilador da el error.

{} LLAVES

Las Llaves son un parte importante del lenguaje de programación C. Se utilizan en diferentes construcciones (ver ejemplos al final), esto a veces puede ser confuso para los principiantes.

Una llave de apertura "{" siempre debe ir seguida de una llave de cierre "}". Esta es una condición a la que se suele referir como llaves emparejadas. El IDE (Entorno Integrado de Desarrollo) Arduino incluye una característica para comprobar si las llaves están emparejadas. Sólo tienes que seleccionar una Llave o incluso hacer click en el punto de inserción que sigue inmediatamente a una llave, y su compañera lógica será seleccionada.

En la actualidad esta característica tiene un pequeño fallo, el IDE encuentra a menudo (incorrectamente), llaves en el texto que pueden estar situadas dentro de comentarios.

Los programadores principiantes y los programadores que llegan a C desde el lenguaje BASIC a menudo encuentran dificultades o grandes confusiones usando llaves. Después de todo, las llaves reemplazan el RETURN en una subrutina(función), el ENDIF en un condicional y el NEXT en un loop FOR.

Dado que el uso de las llaves es tan variado, es una buena práctica de programación escribir la llave de cierre inmediatamente después de escribir la llave de apertura, cuando se inserta una construcción que requiere llaves. Después insertar algunos saltos de línea (líneas en blanco) entre las llaves y comenzar a insertar sentencias. De esta forma tus llaves y su posición, nunca llegarán a estar desemparejadas.

Llaves desemparejadas a menudo pueden conducir a errores de compilación misteriosos y difíciles de comprender, que pueden ser complicados de rastrear en un programa largo. Debido a sus variados usos, las llaves también son increíblemente importantes para la sintaxis de un programa, el movimiento de una llave una o dos líneas, a menudo afecta de manera dramática el significado de un programa.

Usos principales de las Llaves

Funciones

```
void myfunction(tipodato argumento){
  sentencia(s)
}
```

Loops

```
while (expresión booleana)
{
  sentencia(s)
}

do
```

```
  {
    sentencia(s)
} while (expresión booleana);

for (inicialización; condición final; expresión incremento)
{
  sentencia(s)
}
```

Sentencias Condicionales

```
if (expresión booleana)
{
   sentencia(s)
}

else if (expresión booleana)
{
   sentencia(s)
}
else
{
   sentencia(s)
}
```

// COMENTARIOS

Los comentarios son líneas en el programa para aclarar a tí o a otros sobre el funcionamiento del programa. Estas líneas son ignoradas por el compilador y no se exportan al procesador. No ocupan por tanto espacio en el Chip Atmega

El único propósito de los comentarios es que entiendas o entiendan (o recuerdes) cual es el funcionamiento de tu programa. Existen dos maneras distintas de marcar una línea como comentario:

Ejemplo

```
 x = 5;   // Esto es una línea simple de comentario. Todo lo que va
después de la doble barra es un comentario
         // Hasta el final de la línea

/* Esto es un comentario multilínea - úsalo para comentar bloques
enteros de código

if (gwb == 0){   // Una línea de comentario sencilla puede usarse dentro
                  de un comentario multilínea
x = 3;              /* pero no otro comentario multilínea- esto no es
válido */
}
// No olvides cerrar el comentario multilínea
*/
```

Consejo

Cuando estás experimentando con código, "comentar" partes del programa es una útil de eliminar líneas que puedan contener errores. Así dejamos las líneas en el código, pero como comentario , así que serán ignoradas por el compilador. Es especialmente útil cuando estamos intentando localizar el problema , o cuando el compilador rechaza el programa.

/**/ COMENTARIOS

Los comentarios son líneas en el programa para aclarar a tí o a otros sobre el funcionamiento del programa. Estas líneas son ignoradas por el compilador y no se exportan al procesador. No ocupan por tanto espacio en el Chip Atmega

El único propósito de los comentarios es que entiendas o entiendan (o recuerdes) cual es el funcionamiento de tu programa. Existen dos maneras distintas de marcar una línea como comentario:

Ejemplo

```
x = 5;   // Esto es una línea simple de comentario. Todo lo que va
         después de la doble barra es un comentario
        // Hasta el final de la línea

/* Esto es un comentario multilínea - úsalo para comentar bloques
enteros de código

if (gwb == 0){   // Una línea de comentario sencilla puede usarse dentro
                  de un comentario multilínea
x = 3;       /* pero no otro comentario multilínea- esto no es válido */
}
// No olvides cerrar el comentario multilínea
*/
```

Consejo
Cuando estás experimentando con código, "comentar" partes del programa es una útil de eliminar líneas que puedan contener errores. Así dejamos las líneas en el código, pero como comentario , así que serán ignoradas por el compilador. Es especialmente útil cuando estamos intentando localizar el problema , o cuando el compilador rechaza el programa.

OPERADORES ARITMÉTICOS

= OPERADOR DE ASIGNACIÓN (UN SOLO SÍMBOLO DE "IGUAL")

Guarda el valor en la derecha del símbolo "=" dentro de la variable a la izquierda del símbolo "=".

El signo de igualdad "=" en el lenguaje de programación C se llama el operador de asignación. Tiene un significado diferente que en la clase de álgebra en el que se indica una ecuación o igualdad. El operador de asignación le dice al microcontrolador que evalúe cualquier valor o expresión en el lado derecho del signo igual, y lo almacene en la variable a la izquierda del signo igual.

Ejemplo

```
int sensVal;                // declara una variable int llamada sensVal
senVal = analogRead(0);   // guarda el valor (digitalizado) del   voltaje
                           de entrada del pin analógico 0 en SensVal
```

Sugerencias de programación

The variable on the left side of the assignment operator (= sign) needs to be able to hold the value stored in it. If it is not large enough to hold a value, the value stored in the variable will be incorrect.

Don't confuse the assignment operator [=] (single equal sign) with the comparison operator [==] (double equal signs), which evaluates whether two expressions are equal.

La variable en el lado izquierdo del operador de asignación (signo "=") tiene que ser capaz de mantener el valor almacenado en ella. Si no es suficientemente grande para contenerlo, el valor almacenado en la variable será incorrecto.

No confunda el operador de asignación [=] (un solo signo igual) con el operador de comparación [==] (un signo igual doble), que evalúa si dos expresiones son iguales.

SUMA, RESTA, MULTIPLICACIÓN Y DIVISIÓN

Descripción

Estos operadores devuelven la suma, diferencia, producto o cociente (respectivamente) de los dos operandos. La operación se lleva a cabo utilizando el tipo de datos de los operandos, por lo que, por ejemplo, `9 / 4` resulta `2` desde 9 y 4 que son enteros int. Esto también significa que la operación puede desbordarse si el resultado es mayor que el que se puede almacenar en el tipo de datos (por ejemplo, la suma de 1 a un int con el valor de 32.767 resulta -32.768). Si los operandos son de tipos diferentes, se utiliza el tipo del "más grande" para el cálculo.

Si uno de los números (operandos) es del tipo **float** o del tipo **double**, se usará coma flotante para el cálculo.

Ejemplos

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

Sintaxis

```
result = value1 + value2;
result = value1 - value2;
result = value1 * value2;
result = value1 / value2;
```

Parámetros:

value1: cualquier variable o constante

value2: cualquier variable o constante

Sugerencias de programación:

- Debes saber que las integer constants por defecto son int, así que algunos cálculos con constantes pueden provocar desbordamiento (p.e. 60 * 1000 devolverá un resultado negativo)
- Elige los tamaños de variable que sean suficientemente grandes como para alojar el resultado de tus calculos.
- Debes saber en que punto tu variable se desbordará en su máximo, y que esto también ocurre en su mínimo. p.e. (0 - 1) o también (0 - - 32768)
- Para cálculos matemáticos que requieren fracciones, usa variables float, pero ten en cuenta los inconvenientes: gran tamaño, velocidades bajas de cálculo
- Usa el operador de conversión (casting). Por ejemplo: (int)myFloat para convertir el tipo de una variable en el momento.

% (MÓDULO)

Descrición

Calcula el resto de la división entre dos enteros. Es útil para mantener una variable dentro de un rango particular (por ejemplo el tamaño de un array)

Sintaxis

resultado = dividendo % divisor

Parametros

dividendo: el número que se va a dividir

divisor: el número por el que se va a dividir

Devuelve

el resto de la división

Ejemplo

```
x = 7 % 5;   // x ahora contiene 2
x = 9 % 5;   // x ahora contiene 4
x = 5 % 5;   // x ahora contiene 0
x = 4 % 5;   // x ahora contiene 4
```

Código de ejemplo

```
/* actualiza un valor en el array cada vez que se pasa por el bucle */

int valores[10];
int i = 0;

void setup() {}

void loop()
{
  valores[i] = analogRead(0);
  i = (i + 1) % 10;   // el operador módulo prevalece sobre la variable
}
```

Nota

El operador modulo no funciona con datos en coma flotante (float)

OPERADORES BOOLEANOS

Se pueden usar dentro de operaciones condicionales o en una sentencia if.

&& (AND lógico)

Verdadero sólo si ambos operadores son Verdadero, por ejemplo:

```
if (digitalRead(2) == HIGH  && digitalRead(3) == HIGH) { // lee dos pulsadores
  // ...
}
```

Es Verdadero sólo si ambas entradas estás activadas, es decir, en estado HIGH.

|| (OR lógico)

Verdadero si alguno de los dos operadores es Verdadero, por ejemplo:

```
if (x > 0 || y > 0) {
  // ...
```

```
}
```

Es Verdadero si alguno de los valores x ó y es mayor que 0.

! (NOT)

Verdadero si el operador es Falso, por ejempo:

```
if (!x) {
  // ...
}
```

Es Verdadero si el valor de x es Falso (p. e. si x es igual a 0).

Atención.

Asegúrate de no confundir el operador AND booleano && (doble signo &) con el operador AND para bits & (un solo signo &).

De la misma manera no hay que confundir el operador OR booleano || (doble barra vertical) con el operador OR para bits | (una sola barra vertical).

El operador NOT de bits ~ (tilde) tiene una apariencia muy diferente del operador NOT booleano ! (exclamación o "bang", como lo llaman algunos programadores), pero debes asegurarte de usar cada uno de ellos dentro del contexto adecuado.

Ejemplos.

```
if (a >= 10 && a <= 20){}   // Verdadero sólo si el valor de a está entre 10 y
20
```

OPERADORES DE COMPOSICIÓN

++ (INCREMENTO) / -- (DISMINUCIÓN)

Descripción

Incrementa o disminuye una variable

Sintaxis

```
x++;  // incrementa x en uno y regresa el valor anterior de x
++x;  // incrementa x en uno y regresa el nuevo valor de x

x-- ;   // disminuye x en uno y y regresa el valor anterior de x
--x ;   // disminuye n uno y regresa el nuevo valor de x
```

Parámetros

x: un entero o long (puede aceptar sin signo)

Devuelve

El valor original o el resultante del incremento o disminución de la variable.

Ejemplos

```
x = 2;
y = ++x;      // x ahora guarda 3, y guarda 3
y = x--;      // x guarda 2 de nuevo, y sigue guardando 3
```

+= , -= , *= , /=

Descripción

Realiza una operación matemática con una variables con respecto a otra variable o una constante. El operador += (y los demás operadores) son una forma simplificada de la sintaxis completa, tal y como se muestra más abajo.

Sintaxis

```
x += y;   // equivalente a la expresión x = x + y;
x -= y;   // equivalente a la expresión x = x - y;
x *= y;   // equivalente a la expresión x = x * y;
x /= y;   // equivalente a la expresión x = x / y;
```

Parámetros

x: cualquier tipo de variable

y: cualquier tipo de variable o constante

Ejemplos

```
x = 2;
x += 4;      // x ahora es 6
x -= 3;      // x ahora es 3
x *= 10;     // x ahora es 30
x /= 2;      // x ahora es 15
```

2. VARIABLES:

CONSTANTES

Las constantes variables que vienen predefinidas en el lenguaje de Arduino. Se usan para facilitar la lectura de los programas. Clasificamos las constantes en grupos.

Las que definen niveles lógicos, verdadero (true) y falso (false) (Constantes Booleanas)

Existen dos constantes para representar si algo es cierto o falso en Arduino: **true**, y **false**.

FALSE

false es el más sencillo de definir. false se define como 0 (cero).

TRUE

true se define la mayoría de las veces como 1, lo cual es cierto, pero tiene una definición más amplia. Cualquier entero que es *no-cero* es TRUE, en un sentido Booleano. Así, en un sentido Booleano, -1, 2 y -200 son todos true.

Ten en cuenta que las constantes *true* y *false* se escriben en minúsculas, al contrario que HIGH, LOW, INPUT, y OUTPUT.

Las que definen el nivel de los pines, nivel alto (HIGH) y nivel bajo (LOW)

Cuando leemos o escribimos en un pin digital, existen sólo dos valores que podemos obtener o asignar : **HIGH** y **LOW**.

HIGH

El significado de HIGH (en referencia a un pin) depende de si el pin está configurado como entrada (INPUT) o como salida (OUTPUT). Cuando un pin se configura como entrada (INPUT) usando pinMode, y se lee con digitalRead, el microcontrolador nos retornará HIGH si en el pin hay 3 voltios o más.

Un pin puede ser configurado como entrada (INPUT) usando pinMode, y después establecerlo a HIGH con digitalWrite, esto conectará el pin a 5 Voltios a través de una resistencia interna de 20K, resistencia pull-up , la cual establecerá el pin al estado de lectura HIGH a menos que la conectemos a una señal LOW a través de un circuito externo.

Cuando un pin se configura como salida (OUTPUT) con pinMode, y se establece a HIGH con digitalWrite, el pin tiene 5V. Ene ste estado puede usarse como *fuente* de corriente , e.j. Luz y LED que se conectan a través de resistencias en serie a masa (tierra), o a otro pin configurado como salida y establecido a LOW.

LOW

El significado de LOW difiere también según esté configurado como entrada (INPUT) o como salida (OUTPUT). Cuando un pin está configurado como entrada (INPUT) con pinMode, y se lee con digitalRead, el microcontrolador retornará LOW si el voltaje presente en el pin es de 2V o menor.

Cuando un pin es configurado como salida (OUTPUT) con pinMode,y establecido LOW con digitalWrite, el pin tiene 0 voltios. En este estado puede *meter* corriente, e.j. Luz y LED que se conectan a través de resistencias en serie a +5 voltios, o a otro pin configurado como salida, y establecido a HIGH.

Las que definen los pines digitales, INPUT y OUTPUT

Los pines digitales pueden se usados como **entrada (INPUT)** o como **salida (OUTPUT)**. Cambiando un pin de INPUT a OUTPUT con pinMode() el comportamiento eléctrico del pin cambia drásticamente.

Pins configurados como entradas

Los pins de Arduino (Atmega) configurados como **INPUT** con pinMode() se dice que se encuentran en un estado de alta impedancia. Una forma de explicar esto es que un pin configurado como entrada se le aplica una muy baja demanda, es decir una resistencia en serie de 100 Megohms. Esto lo hace muy útil para leer un sensor, pero no para alimentar un LED.

Pins configurados como salidas

Los pins configurados como **salida (OUTPUT)** con pinMode() se dice que estan en estado de baja impedancia. Esto implica que pueden proporcionar una sustancial cantidad de corriente a otros circuitos. Los pins de Atmega pueden alimentar (proveer de corriente positiva) or meter (proveer de masa) hasta 40 mA (miliamperios) de corriente a otros dispositivos/circuitos. Esto o hace muy útil para alimentar LED's pero inservible para leer sensores. Los pins configurados como salida pueden deteriorarse o romperse si ocurre un cortocircuito hacia los 5V o 0V . La cantidad de corriente que puede proveer un pin del Atmega no es suficiente para la mayoría de los relés o motores, y es necesario añadir circuitería extra.

INTEGER CONSTANTS

Integer Constants son números utilizados directamente en un *sketch*, como `123`. Por defecto, éstos números son tratados como int, pero puedes cambiarlo con las letras *U* y *L* (ver abajo).

Normalmente, las constantes *integer* son tratadas como enteros base 10 (decimales), pero se puede utilizar notación especial (formateadores) para ingresar números en otras bases.

Base	Ejemplo	Formateador	Comentario
10 (decimal)	123	Ninguno.	
2 (binario)	B1111011	Antecede "B"	Sólo funciona con valores de 8 bits (0 to 255). Caracteres 0-1 válidos.
8 (octal)	0173	Antecede "0"	Caracteres 0-7 válidos.
16 (hexadecimal)	0x7B	Antecede "0x"	Caracteres 0-9, A-F, a-f válidos.

Decimal es base 10. Esta es la matemática de sentido común con que se conocen. Para constantes sin otros prefijos, se asume el formato decimal.

Ejemplo:

```
101     // igual a 101 decimal   ((1 * 10^2) + (0 * 10^1) + 1)
```

Binary es base dos. Sólo caracteres 0 y 1 son válidos.

Ejemplo:

```
B101    // igual a 5 decimal   ((1 * 2^2) + (0 * 2^1) + 1)
```

El formateador binario sólo funciona en *bytes* (8 *bits*) entre 0 (B0) y 255 (B11111111). Si resulta conveniente ingresar un entero (*int*, 16 *bits*) de forma binaria, puedes hacer un procedimiento de dos pasos, como a continuación:

```
valorInt = (B11001100 * 256) + B10101010;    // B11001100 es el ''byte'' alto.
```

Octal es base ocho. Sólo caracteres de 0 hasta 7 son válidos. Los valores Octales son indicados por el prefijo "0"

Ejemplo:

```
0101    // igual a 65 decimal   ((1 * 8^2) + (0 * 8^1) + 1)
```

Atención
Es posible generar un *bug* o error difícil de encontrar al incluír (de forma involuntaria) un cero antecediento una constante, logrando así que el compilador interprete tu constante como octal.

Hexadecimal (ó hex) es base dieciséis. Los caracteres válidos son del 0 al 9, y las letras desde la *A* hasta la *F*; *A* tiene el valor de 10, *B* es 11, *C* es 12, *D* es 13, *E* es 14, y la F, como ya habrás adivinado, es 15. Los valores hexadecimales se indican con el prefijo "0x". Nota que los valores de la *A* a la *F*, pueden ser escritos en mayúscula o minúscula.

Ejemplo:

```
0x101   // igual a 257 decimal   ((1 * 16^2) + (0 * 16^1) + 1)
```

TIPOS DE DATOS

BOOLEANOS

Un **booleano** sólo puede tomar dos valores, Verdadero o Falso. Cada booleano ocupa un único byte en la memoria.

Ejemplo:

```
int LEDpin = 5;       // LED en el pin 5
int switchPin = 13;   // pulsador en el pin 13, su otra patilla conectada en GND

boolean running = false;  // crea la variable booleana running y le asisga el
valor Falso (false)

void setup()
{
  pinMode(LEDpin, OUTPUT);
  pinMode(switchPin, INPUT);
  digitalWrite(switchPin, HIGH);      // conecta la resistencia pull-up interna
del pin 13
}

void loop()
{
  if (digitalRead(switchPin) == LOW)
  {  // si el pulsador es accionado la pull-up mantiene el pin en estado HIGH
    delay(100);                          // retardo para impedir un rebote en el
pulsador
     running = !running;                    // invierte el valor de la variable
running
    digitalWrite(LEDpin, running)      // enciende el LED
  }
}
```

CHAR

Descripción

Es un tipo de dato que ocupa un byte de memoria y almacena un valor de carácter. Los carácteres literales se escriben con comillas simples: 'A' (para varios carácteres -strings- utiliza dobles comillas "ABC").

De todas maneras los carácteres son almacenados como números. Puedes ver su codificado en la tabla ASCII. Con esto podemos entender que es posible realizar cálculos aritméticos con los carácteres, en este caso se utiliza el valor ASCII del carácter (por ejemplo 'A' + 1 tiene el valor de 66, ya que el valor ASCII de la letra mayúscula A es 65). Mira Serial.println para tener mas información de como son traducidos a números los carácteres.

El tipo de datos char tiene signo. esto significa que codifica números desde -128 hasta 127. Para un dato de un byte (8bits), utiliza el tipo de dato "byte".

Ejemplo

```
  char miChar = 'A';
  char miChar = 65;      // los dos son equivalentes
```

BYTE

Descripción

Un byte almacena un número sin signo de 8-bit, desde 0 hasta 255.

Ejemplo

```
byte b = B10010;  // "B" es el formateador binario (B10010 = 18 decimal)
```

INT

Descripción

Integers (Números enteros) son el principal tipo de datos para almacenar números, y guardan valores de **2 bytes**. Esto produce un rango entre *-32,768* hasta *32,767* (valor mínimo de -2^15 y un valor máximo de (2^15) - 1).

Variables tipo **Int**, almacenan números negativos con una técnica llamada Complemento a dos. El *bit* más alto, a veces llamado como *"sign" bit*, indica que el número es negativo. Se invierte el valor de cada uno de los *bits*, es decir, se realiza el complemento a uno, y se suma 1 al número obtenido.

La placa Arduino, se encarga de tratar con números negativos por tí, para que las operaciones aritméticas trabajen de manera transparente y en la forma esperada. Sin embargo, pueden haber complicaciones inesperadas al tratar con el operador right bitshift (>>).

Ejemplo

```
int ledPin = 13;
```

Syntax

```
int var = val;
```

- var - nombre de la variable *int*.
- val - valor asignado a dicha variable.

Consejo

Cuando las variables son hechas para exceder su límite, éstas vuelven a su capacidad mínima, ésto sucede en ambas direcciones:

```
int x
x = -32,768;
 x = x - 1;        // x ahora contiene 32,767 - vuelve a empezar en dirección
contraria.

x = 32,767;
x = x + 1;       // x ahora contiene -32,768 - vuelve a empezar.
```

UNSIGNED INT

Descripción

Los enteros sin firma (unsigned int) son los mismos enteros de modo que almacenan un valor de dos bytes. En lugar de almacenar números negativos, sólo almacenan valores positivos, generando un rango útil desde 0 a 65,535 (2^16 - 1).

La diferencia entre los enteros sin firma y los enteros (firmados), reside en que el bit más alto, a veces referenciado como el bit "firmado", es interpretado. En el tipo de datod int de Arduino (que es firmado), si el bit más alto es "1", el número es interpretado como un número negtivo, y los otros 15 bits son interpretados con complemento a 2.

Ejemplo

```
unsigned int ledPin = 13;
```

Sintáxis

```
unsigned int var = val;
```

- var - el nombre de tu variable unsigned int
- val - el valor que asignas a a esa variable

Consejo de codificación

Cuando las variables sobrepasan su capacidad máxima dan la vuelta a su mínima capacidad. Ten en cuenta que esto sucede en ambas direcciones.

```
unsigned int x
x = 0;
 x = x - 1;         // x ahora contiene 65535 - da la vuelta en dirección negativa
x = x + 1;       // x ahora contiene 0 - da la vuelta
```

LONG

Descripción

Las variables de tipo Long son variables de tamaño extendido para almacenamiento de números, y 32 bits (4 bytes), desde -2,147,483,648 hasta 2,147,483,647.

Ejemplo

```
long speedOfLight = 186000L;   // ver Constantes Integer para la explicación de la 'L'
```

Sintaxis

```
long var = val;
```

- var - nombre de la variable tipo Long
- val - valor asignado a la variable

UNSIGNED LONG

Descripción

Las variable long sin firmar (unsigned long) son variables extendidas para almacenar números, y almacenar 32 bits (4 bytes). Por el contrario que las variables long estándar, las unsigned long no almacenan números negativos, haciendo que su rango sea de 0 a 4,294,967,295 (2^32 - 1).

Ejemplo

```
unsigned long tiempo;

void setup()
{
  Serial.begin(9600);
}

void loop()
```

```
{
  Serial.print("Tiempo: ");
  tiempo = millis();
  //imprime el tiempo desde el inicio del programa
  Serial.println(tiempo);
  // espera un segundo para no enviar cantida de masivas de datos
  delay(1000);
}
```

Sintáxis

```
unsigned long var = val;
```

- var - el nombre de tu variable long
- val - el valor que asignas a tu variable long

FLOAT

Descripcion

El tipo variable para los números en coma flotante (número decimal). Estos números son usados, habitualmente, para aproximar valores analógicos y continuos, debido a que ofrecen una mayor resolución que los enteros. Las variables tipo **float** tienen el valor máximo 3.4028235E+38, y como mínimo pueden alcanzar el -3.4028235E+38. Ocupan 4bytes (32bits).

Los **floats** tienen una precisión de 6 o 7 dígitos decimales. Esto significa el número total de dígitos, no el número a la derecha de la coma decimal. Al contrario que en otras plataformas, donde tu podrías obtener mayor precisión usando una variable tipo double (por ejemplo, por encima de 15 dígitos), en Arduino los double tienen el mismo tamaño que los float.

Los números en coma flotante no son exactos, y muchos proporcionan falsos resultados cuando son comparados. Por ejemplo, `6.0 / 3.0` puede no ser igual a `2.0`. Debes comprobar que el valor absoluto de la diferencia entre los números pertenezca a un rango pequeño.

La matemática en coma flotante es mucho más lenta que la matemática de enteros para realizar operaciones, por lo que deberías evitarla si, por ejemplo, un bucle tiene que ejecutarse a la máxima velocidad para funciones con temporizaciones precisas. Los programadores normalmente suelen asignar unas longitudes para convertir las operaciones de coma flotante en cálculos con enteros, para aumentar la velocidad.

Ejemplos

```
float myfloat;
float sensorCalbrate = 1.117;
```

Sintaxis

```
float var = val;
```

- **var** - el nombre de la variable tipo float
- **val** - el valor que le asignas a esa variable

Código de ejemplo

```
int x;
int y;
float z;

x = 1;
y = x / 2;            // y ahora contiene 0, la parte entera de la operación
z = (float)x / 2.0;   // z ahora contiene 0.5 (se debe usar 2.0, en lugar de
```

2)

DOUBLE

Descripcion

Número en coma flotante de doble precisión. Ocupa 4 bytes.

La implementación "double" en Arduino es exactamente lo mismo que la FLOAT, sin ganar nada de precisión.

Consejo

Los usuarios que porten código de otras fuentes y que incluyan variable tipo double deberían examinar el código para ver si la precisión necesaria es diferente a la que se puede lograr en Arduino.

STRING

Descripción

Los strings se representan como arrays de caracteres (tipo char) que terminan con el caracter NULL.

Ejemplos

Todas las siguientes son declaraciones válidas de strings.

```
char Str1[15];
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
char Str4[ ] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino";
```

Posibilidades de declaraciónd e strings

- Declarar un array de caracteres sin incializarlo como en Str1
- Declarar un array de caracteres (con un caracter extra) y el compilador añadirá el caracter NULL requerido, como en STR2
- Explicitar el caracter NULL, Str3
- Inicializar con un string constante entre comillas dobles; el compilador medirá el tamaño del array para ajustar el string constante y caracter NULL para finalizar, Str4
- Inicializar el array con un tamaño explícito y un string constante, Str5
- Inicializar el array, dejando un espacio extra para un string más largo, Str6

Terminación NULL

Generalmente, los strings se finalizan con un caracter NULL (código ASCII 0). Esto permite a funciones (como Serial.print()) establecer dónde está el final del string. De otra forma, seguiría leyendo los siguientes bytes de la memoria que no forman parte del string.

Esto significa que tu string necesita tener espacio para un caracter más del texto que quieres que contenga. Esto es por lo que Str2 y Str5 necesitan 8 caracteres, incluso aunque "arduino" tenga sólo 7 - la última posición es automáticamante completada con un caracter NULL. Str4 será automáticamente dimensionada a 8 caracteres, uno para el NULL extra. En Str3, hemos incluido nosotros mismos el caracter NULL (escrito como '\0').

Ten en cuenta que es posible tener un string sin un caracter NULL al final (p.e. si tú has especificado la longitud de Str2 como 7 en lugar de 8). Esto romperá la mayoría de funciones que usen strings, por lo que no deberías hacerlo intencionadamente. Por lo tanto, si detectas algún comportamiento extraño (operando con los caracteres, no con el string), este podría se el problema

Comillas simples o dobles

Los string siempre se definen entre comillas dobles ("Abc") y los caracteres siempre se definen dentro de comillas simples ('A').

Envolviendo string largos

Puedes envolver strings largos de ésta manera:

```
char miString[] = "Esta es la primera linea"
" esta es la segunda linea"
" etcetera";
```

Arrays de strings

Amenudo es conveniente, al trabajar con grandes cantidades de texto, como proyectos con displays LCD, configurar un array de strings. Como los strings son en sí mismo arrays, esto es un ejemplo real de un array bidimensional.

En el código de abajo, el asterisco después del tipo de dato char "char*" indica que es un array de "punteros". Todos los nombres de arrays son punteros, por lo que esto es necesario para crear un array de aarys. Los punteros son una de las partes más exotéricas de C como para que los principieantes puedan llegar a entenderlas, pero no es necesario entender punteros en detalle para usarlos con efectividad.

Ejemplo

```
char* miStrings[]={"Este es el string 1", "Este es el string 2", "Este es el
string 3",
"Este es el string 4", "Este es el string 5","Este es el string 6"};

void setup(){
Serial.begin(9600);
}

void loop(){
for (int i = 0; i < 6; i++){
   Serial.println(miStrings[i]);
   delay(500);
   }
}
```

ARRAYS

Una matriz o "array" es una colección de variables que son accedidas mediante un número de índice. Los "arrays" en el lenguaje de programación C, en el cual está basado Arduino, pueden ser complicadas, pero usar "arrays" simples es relativamente sencillo.

Creando (Declarando) una matriz o Array

Todos los métodos de abajo son formas válidas de crear (declarar) una matriz.

```
  int myInts[6];
  int myPins[] = {2, 4, 8, 3, 6};
  int mySensVals[6] = {2, 4, -8, 3, 2};
  char message[6] = "hola";
```

Puedes declarar una matriz sin inicializarla como en myInts.

En myPins se declara una matriz sin establecer un tamaño explícito. El compilador cuenta el número de elementos y crea la matriz con el tamaño apropiado.

Finalmente, puedes tanto declarar con un tamaño la matriz e inicializarla al mismo tiempo, como en mySensVals

Accediendo a una matriz o Array

Los Arrays son **zero indexed**, esto significa que, al referirse a una matriz, el primer elemento de la matriz está en el indice 0. Por lo tanto:

`mySensVals[0] == 2, mySensVals[1] == 4,` y sucesivos.

Esto también significa que en una matriz con 10 elementos. el indice 9 es el último elemento. Por lo tanto:

```
int myArray[10]={9,3,2,4,3,2,7,8,9,11};
     // myArray[9]    contiene 11
      // myArray[10]    es invalido y contiene información aleatoria (de otra
dirección de memoria)
```

Por esta razón hay que tener cuidado en el acceso a las matrices. El acceso más allá del final de una matriz (usando un número de índice mayor que el tamaño declarado - 1) resultará la lectura de la memoria que está en uso para otros fines. La lectura de estos lugares probablemente no va a hacer mucho mal, excepto la lectura de datos no válidos. Escribir en las localidades de memoria aleatoria es definitivamente una mala idea y, a menudo puede conducir a resultados inesperados como fallos o mal funcionamiento del programa. Esto también puede ser un error difícil encontrar.

A diferencia de BASIC o JAVA, el compilador de C no realiza ninguna comprobación para ver si el acceso a una matriz está dentro de los límites del tamaño de la matriz que ha declarado.

Para asignar un valor a una matriz:

```
mySensVals[0] = 10;
```

To retrieve a value from an array:

```
x = mySensVals[4];
```

Matrices y los bucles FOR

Arrays are often manipulated inside **for** loops, where the loop counter is used as the index for each array element. For example, to print the elements of an array over the serial port, you could do something like this: Las matrices se utilizan muchas veces en el interior de bucles **for**, donde el contador de bucle se utiliza como el índice de cada elemento de la matriz. Por ejemplo, para imprimir los elementos de una matriz a través del puerto serie, se podría hacer algo como esto:

```
int i;
for (i = 0; i < 5; i = i + 1) {
  Serial.println(myPins[i]);
}
```

VOID

La palabra reservada void se usa sólo en la declaración de funciones. Indica que se espera que no devuleva información a la función donde fué llamada.

Ejemplo:

```
// varias acciones se llevan a cabo en las funciones "setup" y "loop"
// pero no se reporta ninguna información al programa principal.

void setup()
{
  // ...
}

void loop()
{
  // ...
```

```
}
```

3. FUNCIONES:

E/S DIGITALES

PINMODE()

Descripción

Configura el pin especificado para comportarse como una entrada o una salida. Mira la descripción de See the description of pines digitales para más información.

Sintaxis

pinMode(pin, modo)

Parametros

pin: el numero del pin que se desea configurar

modo: INPUT (Entrada) o OUTPUT (Salida)

Devuelve

Nada

Ejemplo

```
int ledPin = 13;                 // LED conectado al pin digital 13

void setup()
{
  pinMode(ledPin, OUTPUT);      //configura el pin del LED como salida
}

void loop()
{
  digitalWrite(ledPin, HIGH);   // Enciende el LED
  delay(1000);                  // espera un segundo
  digitalWrite(ledPin, LOW);    // Apaga el LED
  delay(1000);                  // Espera un segundo
}
```

Nota

Una entrada analógica puede ser usada como un pin digital, refiriéndose a ellos desde el número 14 (entrada analógica 0) a 19 (entrada analógica 5).

DIGITALWRITE()

Descripción

Escribe un valor HIGH o LOW hacia un pin digital.

Si el pin ha sido configurado como OUTPUT con pinMode(), su voltaje será establecido al correspondiente valor: 5V

(o 3.3V en tarjetas de 3.3V) para HIGH, 0V (tierra) para LOW.

Si el pin es configurado como INPUT, escribir un valor de HIGH con digitalWrite() habilitará una resistencia interna de 20K conectada en pullup (ver el tutorial de pines digitales). Escribir LOW invalidará la resistencia. La resistencia es sufuciente para hacer brillar un LED de forma opaca, si los LEDs aparentan funcionar, pero no muy iluminados, esta puede ser la causa. La solución es establecer el pin como salida con la función pinMode().

NOTA: El pin digital número 13 es más difícil de usar que los otros pines digitales por que tiene un LED y una resistencia adjuntos, los cuales se encuentran soldados a la tarjeta, y la mayoría de las tarjetas se encuentran así. Si habilitas la resistencia interna en pullup, proporcionará 1.7V en vez de los 5V esperados, por que el LED soldado en la tarjeta y resistencias bajan el nivel de voltaje, significando que siempre regresará LOW. Si debes usar el pin número 13 como entrada digital, usa una resistencia externa conectada a pulldown.

Sintaxis

digitalWrite(pin, valor)

Parameters

pin: el número de pin

valor: HIGH o LOW

Devuelve

nada

Ejemplo

```
int ledPin = 13;                 // LED conectado al pin digital 13

void setup()
{
  pinMode(ledPin, OUTPUT);      // establece el pin digital como salida
}

void loop()
{
  digitalWrite(ledPin, HIGH);   // enciende el LED
  delay(1000);                  // espera por un segundo
  digitalWrite(ledPin, LOW);    // apaga el LED
  delay(1000);                  // espera por un segundo
}
```

Establece el pin número 13 a HIGH, hace un retraso con la duración de un segundo, y regresa el pin a LOW.

Nota

Los pines analogicos pueden ser usados como pines digitales con los números 14 (entrada analogica número 0) hasta 19 (entrada analogica número 5).

DIGITALREAD()

Descripción

Lee el valor de un pin digital especificado, HIGH o LOW.

Sintaxis

digitalRead(pin)

Parámetros

pin: el número de pin digital que quieres leer (*int*)

Devuelve

HIGH o LOW

Ejemplo

```
int ledPin = 13; // LED conecado al pin digital número 13
int inPin = 7;   // botón (pushbutton) conectado al pin digital número 7
int val = 0;     // variable donde se almacena el valor leído

void setup()
{
   pinMode(ledPin, OUTPUT);        // establece el pin digital número 13 como
salida
  pinMode(inPin, INPUT);      // establece el pin digital número 7 como entrada
}

void loop()
{
  val = digitalRead(inPin);   // leer el pin de entrada
  digitalWrite(ledPin, val);    // establece el LED al valor del botón
}
```

Establece el pin número 13 al mismo valor que el pin número 7, que es una entrada.

Nota

Si el pin no esta conectado a algo, digitalRead() puede regresar HIGH o LOW (y esto puede cambiar aleatoriamente).

Los pines analogicos pueden ser usados como pines digitales con los números 14 (entrada analogica número 0) hasta 19 (entrada analogica número 5).

E/S Analógicas

ANALOGREAD()

Descripción

Lee el valor de tensión en el pin analógico especificado. La placa Arduino posee 6 canales (8 canalaes en el Mini y Nano y 16 en el Mega)conectados a un conversor analógico digital de 10 bits. Esto significa que convertirá tensiones entre 0 y 5 voltios a un número entero entre 0 y 1023. Esto proporciona una resolución en la lectura de: 5 voltios / 1024 unidades, es decir, 0.0049 voltios (4.9 mV)por unidad. El rango de entrada puede ser cambiado usando la función analogReference().

El conversor tarda aproximadamente 100 microsegundos (0.0001 segundos) en leer una entrada analógica por lo que se puede llevar una tasa de lectura máxima aproximada de 10.000 lecturas por segundo.

Sintaxis

analogRead(pin)

Parámetros

pin: Indica el número del pin de la entrada analógica que deseamos leer (0 a 5 en la mayoría de las placas, de 0 a 7 en las Mini y Nano y de 0 a 15 en las Mega)

Devuelve

int (0 a 1023)

Nota

Si la entrada analógica que vamos a leer no está conectada a nada, el valor que devolverá la función analogRead() fluctuará dependiendo de muchos valores (los valores de otras entradas analógicas, que tan cerca está tu mano de la entrada en cuestión, etc.)

Ejemplo

```
int analogPin = 3;        // el pin analógico 3 conectado al dial de un
potenciómetro (terminal central del potenciómetro)
                      // los terminales exteriores del potenciómetro conectados
a +5V y masa respectivamente
int val = 0;           // declaración de la variable en la que se almacenará el
valor leído por el conversor.

void setup()
{
  Serial.begin(9600);          //  Inicialización del modulo Serial.
}

void loop()
{
  val = analogRead(analogPin);    // lee el valor de tensión del pin
  Serial.println(val);             // envía el valor leido vía serial.
}
```

ANALOGWRITE()

Descripción

Escribe un valor analógico (PWM) en un pin. Puede ser usado para controlar la luminosidad de un LED o la velocidad de un motor. Después de llamar a la función **analogWrite()**, el pin generará una onda cuadrada estable con el ciclo de trabajo especificado hasta que se vuelva a llamar a la función **analogWrite()** (o una llamada a las funciones **digitalRead()** o **digitalWrite()** en el mismo pin). La frecuencia de la señal PWM sera de aproximadamente 490 Hz.

En la mayoría de las placas Arduino (aquellas con el ATmega168 o ATmega328), se podrá generar señales PWM en los pines 3, 5, 6, 9, 10, y 11. En la placa Arduino Mega, se puede llevar a cabo con los pines desde el 2 hasta el pin 13. Las placas Arduino más antiguas que posean el chip ATmega8 solo podrán usar la función analogWrite() con los pines 9, 10 y 11. No hace faltar configurar el pin como salida para poder usar la función analogWrite().

La función *analogWrite* no tienen ninguna relación con los pines de entrada analógicos ni con la función *analogRead*.

Sintaxis

analogWrite(pin, valor)

Parámetros

pin: Es el pin en el cual se quiere generar la señal PWM.

valor: El ciclo de trabajo deseado comprendido entre 0 (siempre apagado) y 255 (siempre encendido).

Devuelve

Nada

Notas y problemas conocidos.

Las señales PWM generadas en los pines 5 y 6 poseerán ciclos de trabajo superiores a lo esperado. Esto es así por que

para esos dos pines se utiliza el mismo temporizador que se utiliza en las funciones millis() y delay(). Este efecto se notará mucho más en ciclos de trabajo bajos (por ejemplo de 0 a 10) y puede ser que aunque configuremos esos pines con una señal de ciclo de trabajo cero no llegue a ser verdaderamente 0.

Ejemplo

Produce una señal donde conectamos el LED, cuyo ciclo de trabajo es proporcional a la tensión leída en el potenciómetro.

```
int ledPin = 9;      // LED conectado al pin digital 9
int analogPin = 3;   // potenciómetro conectado al pin 3
int val = 0;         // variable en el que se almacena el dato leído

void setup()
{
  pinMode(ledPin, OUTPUT);   // sets the pin as output
}

void loop()
{
  val = analogRead(analogPin);   // lee la tensión en el pin
  analogWrite(ledPin, val / 4);  // los valores de analogRead van desde 0 a 1023
y los valores de analogWrite values van desde 0 a 255, por eso ajustamos el
ciclo de trabajo a el valor leído dividido por 4.
}
```

ANALOGWRITE()

Descripción

Escribe un valor analógico (PWM) en un pin. Puede ser usado para controlar la luminosidad de un LED o la velocidad de un motor. Después de llamar a la función **analogWrite()**, el pin generará una onda cuadrada estable con el ciclo de trabajo especificado hasta que se vuelva a llamar a la función **analogWrite()** (o una llamada a las funciones **digitalRead()** o **digitalWrite()** en el mismo pin). La frecuencia de la señal PWM sera de aproximadamente 490 Hz.

En la mayoría de las placas Arduino (aquellas con el ATmega168 o ATmega328), se podrá generar señales PWM en los pines 3, 5, 6, 9, 10, y 11. En la placa Arduino Mega, se puede llevar a cabo con los pines desde el 2 hasta el pin 13. Las placas Arduino más antiguas que posean el chip ATmega8 solo podrán usar la función analogWrite() con los pines 9, 10 y 11. No hace faltar configurar el pin como salida para poder usar la función analogWrite().

La función *analogWrite* no tienen ninguna relación con los pines de entrada analógicos ni con la función *analogRead*.

Sintaxis

analogWrite(pin, valor)

Parámetros

pin: Es el pin en el cual se quiere generar la señal PWM.

valor: El ciclo de trabajo deseado comprendido entre 0 (siempre apagado) y 255 (siempre encendido).

Devuelve

Nada

Notas y problemas conocidos.

Las señales PWM generadas en los pines 5 y 6 poseerán ciclos de trabajo superiores a lo esperado. Esto es así por que para esos dos pines se utiliza el mismo temporizador que se utiliza en las funciones millis() y delay(). Este efecto se notará mucho más en ciclos de trabajo bajos (por ejemplo de 0 a 10) y puede ser que aunque configuremos esos pines con una señal de ciclo de trabajo cero no llegue a ser verdaderamente 0.

Ejemplo

Produce una señal donde conectamos el LED, cuyo ciclo de trabajo es proporcional a la tensión leída en el potenciómetro.

```
int ledPin = 9;      // LED conectado al pin digital 9
int analogPin = 3;   // potenciómetro conectado al pin 3
int val = 0;         // variable en el que se almacena el dato leído

void setup()
{
  pinMode(ledPin, OUTPUT);   // sets the pin as output
}

void loop()
{
  val = analogRead(analogPin);   // lee la tensión en el pin
  analogWrite(ledPin, val / 4);  // los valores de analogRead van desde 0 a 1023
y los valores de analogWrite values van desde 0 a 255, por eso ajustamos el
ciclo de trabajo a el valor leído dividido por 4.
}
```

E/S AVANZADAS

TONE()

Descripción

Genera una onda cuadrada de la frecuencia especificada (y un 50% de ciclo de trabajo) en un pin. La duración puede ser especificada, en caso contrario la onda continua hasta que haya una llamada a noTone(). El pin puede conectarse a un zumbador piezoeléctrico u otro altavoz que haga sonar los tonos.

Sólo puede generarse un tono cada vez. Si un tono está sonando en un pin diferente, la llamada a tone() no tendrá efecto. Si el tono está sonando en el mismo pin, la llamada establecerá la nueva frecuencia.

El uso de la función tone() interferirá con la salida PWM en los ins 3 y 11 (en otras placas distintas de la Mega).

NOTA: si quieres hacer sonar diferentes tonos en múltiples pins necesitas llamar a noTone() en un pin antes de llamar a tone() en el siguiente pin.

Sintáxis

tone(pin, frecuencia)
tone(pin, frecuencia, duracion)

Parámetros

pin: el pin en el que generar el tono

frecuencia: la frecuencia del tono en hercios.

duracion: la duración del tono en milisegundos (opcional)

NOTA: las frecuencias audibles por el oído humano van de los 20Hz a los 20KHz por lo que el parámetro "frecuencia" debería estar comprendido entre estos dos valores.

Devuelve

Nada

NOTONE()

Descripción

Detiene la generación de la señal cuadrada que se activa al hacer uso de la función tone(). No tiene efecto si no se está generando ningún tono.

NOTA: si quieres generar tonos diferentes en múltiples pines , necesitas usar la función noTone() en el pin antes de llamar a la función tone() en el siguiente pin.

Sintaxis

noTone(pin)

Parámetros

pin: el pin en el cual se va a parar de generar el tono.

Devuelve

Nada

SHIFTOUT()

Descripción

Desplaza un byte de datos bit a bit. Empieza desde el bit más significante (el de más a la izquierda) o el menos significante (el más a la derecha). Cada bit se escribe siguiendo su turno en un pin de datos, después de conmutar un pin de reloj (señal de reloj) para indicar que el bit está disponible.

Esto es conocido como protocolo serie síncrono y es la forma común que utilizan los microcontroladores para comunicarse con sensores y con otros microcontroladores. Ambos dispositivos permanecen sincronizados, y se comunican a velocidades cercanas a las máximas, hasta que ambos compartan la misma linea de reloj. En la documentación del harware interno de los chips se hace referencia a menudo a esta característica como Serial Peripheral Interface (SPI).

Sintaxis

shiftOut(pinDatos, pinReloj, ordenBits, valor)

Parametros

pinDatos: el pin en el cual extraer cada bit (*int*)

pinReloj: el pin que hay que conmutar cada vez que a un **pinDatos** le ha sido enviado el valor correcto (**int**)

ordenBits: en qué orden desplazar los bits; si hacia el **MSBFIRST** (bit más significante primero) o hacia el **LSBFIRST** (bit menos significante primero).
valor: los datos que rotar. (*byte*)

Retorno

Ninguno

Nota

El **pinDatos** y **pinReloj** deben estar ya configurados como salida con una llamada previa a pinMode().

shiftOut se encuentra actualmente configurada para extraer un byte (8 bits) por lo que necesita realizar una operación de dos pasos para extraer valores más grandes de 255.

```
// Haz esto para una comunicación serie MSBFIRST (primero el bit más significativo)
int datos = 500;
// rota el byte más alto
```

```
shiftOut(pinDatos, pinReloj, MSBFIRST, (datos >> 8));
// rota el byte más bajo
shiftOut(datos, pinReloj, MSBFIRST, datos);

// O haz esto para una comunicación serie LSBFIRST (primero el bit menos
significativo)
datos = 500;
// rota el byte más bajo
shiftOut(pinDatos, pinReloj, LSBFIRST, datos);
// rota el bit más alto
shiftOut(pinDatos, pinReloj, LSBFIRST, (datos >> 8));
```

Ejemplo

Para ver el circuito asociado a este ejemplo, ver el tutorial para controlar un registro de desplazamiento 74HC595.

```
//*************************************************************//
//  Name    : shiftOutCode, Hello World                        //
//  Author  : Carlyn Maw,Tom Igoe                              //
//  Date    : 25 Oct, 2006                                     //
//  Version : 1.0                                              //
//  Notes   : Código para utilizar un registro de desplazamiento//
//          :  74HC595 para contar de 0 a 255                  //
//*************************************************************

//Pin conectado al pin ST_CP del 74HC595
int latchPin = 8;
//Pin connectado al pin SH_CP del 74HC595
int clockPin = 12;
////Pin conectado al pin DS del 74HC595
int dataPin = 11;

void setup() {
   // Configura como salida los pines que se direccionan en el bucle principal
(loop)
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
  //rutina de conteo
  for (int j = 0; j < 256; j++) {
     //pone a nivel bajo el latchPin y lo mantienen a nivel bajo todo el tiempo
que estés transmitindo
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, LSBFIRST, j);
    //vuelve a poner el latchPin a nivel alto para señalizar que
    //no sigue siendo necesario eschuchar más información
    digitalWrite(latchPin, HIGH);
    delay(1000);
  }
}
```

PULSEIN()

Descripción

Lee un pulso (ya sea HIGH —alto— o LOW —bajo—) en un pin. Por ejemplo, si **value** es **HIGH**, **pulseIn()** espera a que el pin sea **HIGH**, empieza a cronometrar, espera a que el pin sea **LOW** y entonces detiene la medida de tiempo.

Devuelve la anchura del pulso en microsegundos. Interrumpe la medida y devuelve 0 si el pulso no ha comenzado en un tiempo especificado.

La medida del tiempo en esta función ha sido determinada de forma empírica y está sujeta a errores en pulsos largos. Funciona correctamente en pulsos con una anchura de 10 microsegundos a tres minutos.

Sintaxis

pulseIn(pin, value)
pulseIn(pin, value, timeout)

Parámetros

pin: el número del pin en el que se realiza la medida del pulso. (*int*)

value: tipo de pulso. Puede ser HIGH o LOW. (*int*)

timeout (opcional): el tiempo en microsegundos máximo a esperar antes de que se inicie el pulso. (*unsigned long*)

Devuelve

el ancho del pulso (en microsegundos) ó 0 si el pulso no ha empezado antes del timeout (*unsigned long*)

Ejemplo

```
int pin = 7;
unsigned long duracion;

void setup()
{
  pinMode(pin, INPUT);
}

void loop()
{
  duracion = pulseIn(pin, HIGH);
}
```

TIEMPO

MIN(X, Y)

Descripción

Calcula el mínimo de dos números.

Parametros

x: el primer número, cualquier tipo de dato

y: el segundo número, cualquier tipo de dato

Devuelve

El más pequeño entre los dos números.

Ejemplos

```
sensVal = min(sensVal, 100); // asigna sensVal al menor entre sensVal y 100
                             // asegurando que nunca pasará de 100.
```

Nota

La función max() es usualmente usada para limitar el límite inferior del rango de una variable, mientras que la función min() es usada para limitar el límite superior del rango de una variable.

Advertencia

Debido a la forma en la que se ha implementado la función min(), evite realizar operaciones en el argumentos de la función ya que podría producir errores.

```
min(a++, 100);   // evite esto - puede producir errores

a++;
min(a, 100);     // use este método en cambio - mantenga cualquier operación
fuera de los paréntesis
```

MAX(X, Y)

Descripción

Calcula el máximo de dos números.

Parámetros

x: El primer número, cualquier tipo de dato.

y: El segundo número, cualquier tipo de dato.

Devuelve

El mayor de ambos parámetros.

Ejemplo

```
sensVal = max(senVal, 20); // asigna a sensVal su propio valor o, de ser
superior, 20.
                           // (una forma efectiva de asegurarse que el valor
mínimo de senVal sea 20)
```

Nota

max() suele ser usado para restringir el valor más bajo del rango de una variable, mientras que *min()* es utilizado para restringir el valor máximo del rango.

Atención

Debido a la forma en que la función *max()* es implementada, debes evitar usar otras funciones al definir los parámetros, puede derivar en resultados incorrectos.

```
max(a--, 0);   // evitar esto - puede dar resultados incorrectos.

a--;           // en su lugar, hacerlo así -
max(a, 0);     // mantener cualquier operación fuera de los paréntesis.
```

ABS(X)

Descripción

Calcula el valor absoluto de un número.

Parámetros

x: El numero cuyo valor absoluto deseamos calcular

Devuelve

x: si **x** is mayor o igual que 0.

-x: si **x** es menor que 0.

Precaución

Debido a la forma en la que se ha implementado la función abs(), evite usar otras funciones como parámetro debido a que puede ocasionar que se devuelva un resultado incorrecto.

```
abs(a++);   // evite esto - produce errores en el resultado

a++;          // hazlo de esta manera -
abs(a);       // mantenga cualquier otra operación fuera de la función
```

CONSTRAIN(X, A, B)

Descripción

Restringe un número a un rango definido.

Parámetros

x: el número a restringir, cualquier tipo de datos.

a: El número menor del rango, cualquier tipo de datos.

b: El número mayor del rango, cualquier tipo de datos.

Retorna

x: si **x** está entre **a** y **b**

a: si **x** es menor que **a**

b: si **x** es mayor que **b**

Ejemplo

```
sensVal = constrain(sensVal, 10, 150);
// limita el valor del sensor entre 10 y 150
```

MAP(VALUE, FROMLOW, FROMHIGH, TOLOW, TOHIGH)

Descripción

Re-mapea un número desde un rango hacia otro. Ésto significa que, un valor (*value*) con respecto al rango *fromLow-fromHight* será mapeado al rango *toLow-toHigh*.

No se limitan los valores dentro del rango, ya que los valores *fuera de rango* son a veces objetivos y útiles. La función **constrain()** puede ser usada tanto antes como después de ésta función, si los límites de los rangos son deseados.

Ten en cuenta que los límites "inferiores" de algún rango pueden ser mayores o menores que el límite "superior" por lo

que *map()* puede utilizarse para revertir una serie de números, por ejemplo:

```
y = map(x, 1, 50, 50, 1);
```

La función maneja correctamente también los números negativos, por ejemplo:

```
y = map(x, 1, 50, 50, -100);
```

también es válido y funciona correctamente.

La función *map()* usa matemática de enteros por lo que no generará fracciones, aunque fuere el resultado correcto. Los resultados en fracciones se truncan, y no son redondeados o promediados.

Parámetros

value: el número (valor) a mapear.

fromLow: el límite inferior del rango actual del valor.

fromHigh: el límite superior del rango actual del valor.

toLow: límite inferior del rango deseado.

toHigh: límite superior del rango deseado.

Devuelve

El valor mapeado.

Ejemplo

```
/* Mapea un valor análogo a 8 bits (0 a 255) */
void setup() {}

void loop()
{
  int val = analogRead(0);
  val = map(val, 0, 1023, 0, 255);
  analogWrite(9, val);
}
```

Apéndice

Para los interesados en el funcionamiento de la función, aquí está su código:

```
long map(long x, long in_min, long in_max, long out_min, long out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

POW(BASE, EXPONENTE)

Descripción

Calcula el valor de un número elevado a otro número. Pow() puede ser usado para elevar un número a una fracción. Esta función es útil para generar datos exponenciales o curvas.

Parámetros

base: base que queremos elevar (*float*) (Coma flotante)

exponente: la potencia a la que se quiere elevar la base (*float*) (Coma flotante)

Devuelve

El resultado de la exponenciación (*double*)(Doble)

Ejemplo

Mira la función fscale en la librería del código.

SQ(X)

Descripción

Calcula el cuadrado de un numero: el numero multiplicado por el mismo.

Parámetros

x: el numero, cualquier tipo de dato

Devuelve

El cuadrado del numero

SQRT(X)

Descripción

Calcula la raíz cuadrada de un numero.

Parámetros

x: el numero, cualquier tipo de dato

Devuelve

doble, la raíz cuadrada del numero.

Trigonometría

SIN(RAD)

Descripción

Calcula el seno de un ángulo (en raianes). El resultado será entre -1 y 1.

Parametros

rad: el ángulo en radianes (*float*)

Retorno

El seno del ángulo (*double*)

Nota

Serial.print() y Serial.println() no soportan la impresión de valores de tipo *float*.

COS(RAD)

Descripción

Calcula el coseno de un ángulo (en radianes). El resultado estará entre -1 y 1.

Parámetros

rad: el ángulo en radianes (*float*)

Retorna

El coseno del ángulo ("double")

Nota

Serial.print() y Serial.println() no soportan actualmente la impresión de números con decimales.

TAN(RAD)

Descripción

Calcula la tangente de un ángulo (en radianes). El resultado estará entre el menos infinito y el infinito.

Parámetros

rad: el ángulo en radianes (*float*)

Retorno

La tangente del ángulo (*double*)

Nota

Serial.print() y Serial.println() no soportan actualmente imprimir variales de tipo float.

Números Aleatorios

RANDOMSEED(SEED)

Descripción

randomSeed() inicializa el generador de números pseudoaleatorios, haciéndole empezar en un punto arbitrario de su secuencia aleatoria. Esta secuencia, aunque muy larga y aleatoria, es siempre la misma.

Si es importante que la secuencia de valores generada por random() difiera en ejecuciones sucesivas de un programa, es recomendable utilizar randomSeed() (*seed* en inglés, semilla) para inicializar el generador de números aleatorios con una entrada mínimamente aleatoria como analogRead() en un pin desconectado.

No obstante, puede ser útil usar secuencias pseudoaleatorias que se repitan exactamente. Esto se consigue llamando a randomSeed() con un número fijo antes de empezar la generación de la secuencia.

Parámetros

long, int - recibe un número para generar la semilla.

Devuelve

no devuelve nada

Example

```
long numAleatorio;

void setup(){
  Serial.begin(9600);
  randomSeed(analogRead(0));
}

void loop(){
  randNumber = random(300);
```

```
  Serial.println(numAleatorio);

  delay(50);
}
```

RANDOM()

Descripción

La función *random* genera números pseudoaleatorios.

Sintaxis

random(max)
random(min, max)

Parámetros

min - límite inferior del valor aleatorio, inclusive *(opcional)*

max - límite superior del valor aleatorio, exclusive (se devuelve hasta el anterior)

Devuelve

un número aleatorio entre min y max (*long*)

Nota:

Si es importante que la secuencia de valores generada por random() difiera en ejecuciones sucesivas de un programa, es recomendable utilizar randomSeed() (*seed* en inglés, semilla) para inicializar el generador de números aleatorios con una entrada mínimamente aleatoria como analogRead() en un pin desconectado.

No obstante, puede ser útil usar secuencias pseudoaleatorias que se repitan exactamente. Esto se consigue llamando a randomSeed() con un número fijo antes de empezar la generación de la secuencia.

Ejemplo

```
long numAleatorio;

void setup(){
  Serial.begin(9600);
  // si la entrada analógica 0 no está conectada,
  // la llamada a randomSeed() recibirá ruido estático
  // (analógico) y se generarán diferentes semillas
  // cada vez que se ejecute el sketch.
  randomSeed(analogRead(0));
}

void loop() {
  // escribe un número aleatorio de 0 a 299
  numAleatorio = random(300);
  Serial.println(numAleatorio);

  // escribe un número aleatorio de 10 a 19
  numAleatorio = random(10, 20);
  Serial.println(numAleatorio);

  delay(50);
}
```

COMUNICACIÓN

SERIAL

Se utiliza para la comunicación entre la placa Arduino y un ordenador u otros dispositivos. Todas las placas Arduino tienen al menos un puerto serie (también conocido como UART o USART): **Serial**. Se comunica a través de los pines digitales 0 (RX) y 1 (TX), así como con el ordenador mediante USB. Por lo tanto, si utilizas estas funciones, no puedes usar los pines 0 y 1 como entrada o salida digital.

Puedes utilizar el monitor del puerto serie incorporado en el entorno Arduino para comunicarte con la placa Arduino. Haz clic en el botón del monitor de puerto serie en la barra de herramientas y selecciona la misma velocidad en baudios utilizada en la llamada a begin().

La placa Arduino Mega tiene tres puertos adicionales de serie: **Serial1** en los pines 19 (RX) y 18 (TX), **Serial2** en los pines 17 (RX) y 16 (TX), **Serial3** en los pines 15 (RX) y 14 (TX). Para utilizar estos pines para comunicarse con el ordenador personal, necesitarás un adaptador USB adicional a serie, ya que no están conectados al adaptador USB-Serie de la placa Arduino Mega. Para usarlos para comunicarse con un dispositivo serie externo TTL, conecta el pin TX al pin RX del dispositivo, el RX al pin TX del dispositivo, y el GND de tu Arduino Mega a masa del dispositivo. (No conectes estos pines directamente a un puerto serie RS232, que operan a +/- 12V y esto puede dañar la placa Arduino.)

Funciones

- begin()
- end()
- available()
- read()
- flush()
- print()
- println()
- write()

Funciones

BEGIN()

Descripción

Establece la velocidad de datos en bits por segundo (baudios) para la transmisión de datos en serie. Para comunicarse con el computador, utilice una de estas velocidades: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200. Sin embargo, puedes especificar otras velocidades - por ejemplo, para comunicarte a través de los pines 0 y 1 con un componente que requiere una velocidad de transmisión en particular.

Sintaxis

Serial.begin(speed)

Solamente en Arduino Mega:
Serial1.begin(speed)
Serial2.begin(speed)
Serial3.begin(speed)

Parámetros

speed: Velocidad en bits por segundo (baudios) - *long*

Devuelve

nada

Ejemplo:

```
void setup() {
        Serial.begin(9600);     // abre el puerto serie y establece la velocidad
en 9600 bps
}

void loop() {}
```

Ejemplo para Arduino Mega:

```
// Arduino Mega usando sus 4 puertos serie
// (Serial, Serial1, Serial2, Serial3),
// con diferentes velocidades de datos:

void setup(){
  Serial.begin(9600);
  Serial1.begin(38400);
  Serial2.begin(19200);
  Serial3.begin(4800);

  Serial.println("Hola ordenador");
  Serial1.println("Hola Serial 1");
  Serial2.println("Hola Serial 2");
  Serial3.println("Hola Serial 3");
}

void loop() {}
```

END()

Descripción

Desactiva la comunicación serie, permitiendo a los pines RX and TX ser usados como entradas o salidas digitales. Para reactivar la comunicación serie, llama al método Serial.begin().

Sintaxis

Serial.end()

Solamente en Arduino Mega:

Serial1.end()
Serial2.end()
Serial3.end()

Parámetros

ninguno

Devuelve

nada

AVAILABLE()

Descripción

Devuelve el número de bytes (caracteres) disponibles para ser leídos por el puerto serie. Se refiere a datos ya recibidos y disponibles en el buffer de recepción del puerto (que tiene una capacidad de 128 bytes).

Sintaxis

Serial.available()

Sólo para Arduino Mega:

Serial1.available()
Serial2.available()
Serial3.available()

Parametros

ninguno

Devuelve

el número de bytes disponibles para ser leídos

Ejemplo

```
int incomingByte = 0;   // para los datos de entrada serie

void setup() {
        Serial.begin(9600);     // abre el puerto serie, establece la velocidad
a 9600 bps
}

void loop() {

        // envía datos solo cuando recibe datos:
        if (Serial.available() > 0) {
                // lee el byte de entrada:
                incomingByte = Serial.read();

                // muestra lo que tiene:
                Serial.print("He recibido: ");
                Serial.println(incomingByte, DEC);
        }
}
```

Ejemplo Arduino Mega:

```
void setup() {
  Serial.begin(9600);
  Serial1.begin(9600);

}

void loop() {
  // lee desde el puerto 0, envía al puerto 1:
  if (Serial.available()) {
    int inByte = Serial.read();
    Serial1.print(inByte, BYTE);

  }
  // lee del puerto 1, envía al puerto 0:
  if (Serial1.available()) {
    int inByte = Serial1.read();
    Serial.print(inByte, BYTE);
  }
}
```

READ()

Descripción

Lee los datos entrantes del puerto serie.

Sintaxis

Serial.read()

Solamente en Arduino Mega:

Serial1.read()
Serial2.read()
Serial3.read()

Parámetros

Ninguno

Devuelve

el primer byte disponible recibido por el puerto serie (devuelve -1 si no hay datos disponibles) - *int*

Ejemplo

```
int incomingByte = 0;   // para el byte leido

void setup() {
        Serial.begin(9600);     // abre el puerto serie a 9600 bps
}

void loop() {

        // envia datos solamente cuando recibe datos
        if (Serial.available() > 0) {
                // lee el byte entrante:
                incomingByte = Serial.read();

                // dice lo que ha recibido:
                Serial.print("He recibido: ");
                Serial.println(incomingByte, DEC);
        }
}
```

FLUSH()

Descripción

Vacía el búfer de entrada de datos en serie. Es decir, cualquier llamada a Serial.read () o Serial.available () devolverá sólo los datos recibidos después la llamada más reciente a Serial.flush ().

Sintaxis

Serial.flush()

Solamente en Arduino Mega:

Serial1.flush()
Serial2.flush()
Serial3.flush()

Parámetros

ninguno

Retorna

nada

PRINT()

Descripción

Imprime los datos al puerto serie como texto ASCII. Este comando puede tomar muchas formas. Los números son impresos mediante un juego de caracteres ASCII para cada dígito. Los valores de tipo "float" son impresos en forma de dígitos ASCII con dos decimales por defecto. Los valores tipo "byte" se envían como un único carácter. Los caracteres y las cadenas se envían tal cual. Por ejemplo:

- Serial.print(78) imprime "78"
- Serial.print(1.23456) imprime "1.23"
- Serial.print(byte(78)) imprime "N" (cuyo código ASCII es 78)
- Serial.print('N') imprime "N"
- Serial.print("Hello world.") imprime "Hello world."

Un segundo parámetro opcional especifica la base (formato) a usar; los valores permitidos son BYTE, BIN (binarios o base 2), OCT (octales o base 8), DEC (decimales o base 10), HEX (hexadecimales o base 16). Para números de coma flotante, este parámetro especifica el numero de posiciones decimales a usar. Por ejemplo:

- Serial.print(78, BYTE) imprime "N"
- Serial.print(78, BIN) imprime "1001110"
- Serial.print(78, OCT) imprime "116"
- Serial.print(78, DEC) imprime "78"
- Serial.print(78, HEX) imprime "4E"
- Serial.println(1.23456, 0) imprime "1"
- Serial.println(1.23456, 2) imprime "1.23"
- Serial.println(1.23456, 4) imprime "1.2346"

Sintaxis

Serial.print(val)
Serial.print(val, format)

Parámetros

val: el valor a imprimir - de cualquier tipo

format: especifica el número de la base (para números enteros) o el número de posiciones decimales (para números de coma flotante o tipo "float")

Devuelve

Nada

Ejemplo:

```
/*Usa un bucle FOR para los datos e imprime un número en varios formatos.*/
int x = 0;    // variable

void setup() {
  Serial.begin(9600);      // abre el puerto serie a 9600 bps:
}
```

```
void loop() {
  // print labels
  Serial.print("SIN FORMATO");       // imprime un texto
  Serial.print("\t");              // imprime un tabulado

  Serial.print("DEC");
  Serial.print("\t");

  Serial.print("HEX");
  Serial.print("\t");

  Serial.print("OCT");
  Serial.print("\t");

  Serial.print("BIN");
  Serial.print("\t");

  Serial.println("BYTE");

  for(x=0; x< 64; x++){    // solo una parte de la tabla

    // imprime en varios formatos:
    Serial.print(x);       // imprime como codificado ASCII decimal - igual que
"DEC"
    Serial.print("\t");    // imprime un tabulado

    Serial.print(x, DEC);  // imprime como codificado ASCII decimal
    Serial.print("\t");    // imprime un tabulado

    Serial.print(x, HEX);  // imprime como codificado ASCII hexadecimal
    Serial.print("\t");    // imprime un tabulado

    Serial.print(x, OCT);  // imprime como codificado ASCII octal
    Serial.print("\t");    // imprime un tabulado

    Serial.print(x, BIN);  // imprime como codificado ASCII binario
    Serial.print("\t");    // imprime un tabulado

    Serial.println(x, BYTE);    // imprime el valor en bruto del byte,
    //                             y añade el salto de línea con "println"
    delay(200);            // espera 200 milisegundos
  }
  Serial.println("");      //imprime otro salto de línea
}
```

Sugerencias de programación / Problemas conocidos

El último carácter a imprimir se transmite a través del puerto serie después de que Serial.print () ha regresado.

PRINTLN()

Descripción

Imprime los datos al puerto serie como texto ASCII seguido de un retorno de carro (ASCII 13, o '\r') y un carácter de avance de línea (ASCII 10, o '\n'). Este comando tiene la misma forma que Serial.print ().

Sintaxis

Serial.println(val)
Serial.println(val, format)

Parámetros

val: el valor a imprimir - de cualquier tipo

format: especifica el número de la base (para números enteros) o el número de posiciones decimales (para números de coma flotante o tipo "float")

Devuelve

Nada

Ejemplo:

```
/*
  Analog input

 Lee el pin analógico 0, e imprime su valor por el puerto serie.

 created 24 March 2006
 by Tom Igoe
 */

int analogValue = 0;    // variable para guardar el valor analogico

void setup() {
  // abre el puerto serie a 9600 bps:
  Serial.begin(9600);
}

void loop() {
  // lee la entrada analogica en el pin 0:
  analogValue = analogRead(0);

  // imprime el valor en varios formatos:
  Serial.println(analogValue);       // imprime como ASCII decimal
  Serial.println(analogValue, DEC);  // imprime como ASCII decimal
  Serial.println(analogValue, HEX);  // imprime como ASCII hexadecimal
  Serial.println(analogValue, OCT);  // imprime como ASCII octal
  Serial.println(analogValue, BIN);  // imprime como ASCII binario
  Serial.println(analogValue, BYTE); // imprime el valor del byte

  // espera 10 milisegundos antes de la siguiente lectura.
  delay(10);
}
```

WRITE()

Descripción

Escribe datos binarios en el puerto serie. Estos datos se envían como un byte o una serie de bytes; para enviar los caracteres que representan los dígitos de un número usar función print () en su lugar.

Syntax

Serial.write(val)
Serial.write(str)
Serial.write(buf, len)

Arduino Mega tambien soporta: Serial1, Serial2, Serial3 (en lugar de Serial)

Parámetros

val: un valor para enviar como un solo byte

str: una cadena 'string' para enviar como una serie de bytes

buf: un 'array' para enviar como una serie de bytes

len: longitud del búfer